
**Escuela Politécnica Superior**

**Departamento de Tecnología Electrónica y de las Comunicaciones**

# EXPLORANDO EL PROCESO DE RECOLECCIÓN, ANÁLISIS Y VISUALIZACIÓN DEL TRÁFICO EN LAS REDES DE COMPUTADORAS

## TESIS DOCTORAL

**Carlos Gonzalo Vega Moreno**

**Madrid, Noviembre 2017**

# Explorando el proceso de recolección, análisis y visualización del tráfico en las redes de computadoras


Por

CARLOS GONZALO VEGA MORENO




UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Tecnología Electrónica y de las Comunicaciones


TESIS DOCTORAL

NOVIEMBRE 2017


Tutor: Javier Aracil Rico

# Agradecimientos

Esta tesis es el producto del trabajo realizado durante los últimos años, pero no lo hubiera podido llevar a cabo sin la ayuda y apoyo de muchas personas que siempre estuvieron a mi lado iluminando el camino cuando más oscuro estaba.

En primer lugar, gracias a mi director Javier Aracil, por guiarme estos años y enseñarme las muchas facultades que he aprendido durante la realización de este trabajo, las cuales han sido esenciales para mi desarrollo como investigador y profesional. Gracias por tu tiempo y dedicación.

Las contribuciones de este trabajo no son fruto de una única persona, ni de dos. Detrás hay un grupo de investigación, *High Performance Computing and Networking*, que ha demostrado siempre un gran trabajo en equipo y compañerismo. Gracias a todos por vuestra ayuda y consejos. Quería agradecer en particular la ayuda en el desarrollo de las aportaciones presentadas en esta tesis a Paula, de las primeras personas que conocí al entrar en esta universidad. Gracias a ti y a Rafa por los trabajos que hemos realizado juntos. También quería agradecer a José Fernando, por su ayuda con la generadora de tráfico y sus siempre útiles consejos. Gracias a Mario, por su interés, preocupación durante estos años. Ha sido un placer también trabajar junto con Eduardo, Guillermo y Daniel, que me han enseñando muchísimo durante estos años en los proyectos que hemos trabajado. También quería agradecer a David, Javier, José Luis y a todos los demás compañeros del laboratorio y a los que ya no están, por sus consejos y ayuda.

Trabajar en este grupo de investigación me ha brindado la oportunidad de trabajar con grandes profesionales. Por eso quería agradecer a los profesores del grupo: Sergio, Jorge, Iván, Paco, Gustavo y a todos por permitirme trabajar con vosotros durante todos estos años en los proyectos del grupo. También quería dar las gracias a los trabajadores de Naudit: Felix, Iria, Elisa, Nicolás, y todos los demás por su ayuda en mi realización profesional. Todo este trabajo no hubiera sido posible sin vuestra colaboración.

Gracias también a Juan Carlos Torrado y Javier Gomez del departamento de informática por su apoyo, consejos y los buenos momentos que hemos pasado estos años. Hace ya ocho años que empezamos la carrera Juan Carlos! Gracias por tu constante ayuda (y paciencia), estoy seguro de que tendremos oportunidad de hacer más viajes juntos, de momento ya van tres continentes. Gracias a todos los compañeros y amigos de la carrera, a Sofia (y la familia pepa!), Bartosz, Adalberto, Luis, Roberto, y todos los demás por los grandes momentos que hemos pasado juntos y su apoyo e interés todos estos años. Gracias a los amigos, a Isaac por estar presente desde parvulitos, a Victor y Nacho por todos estos años de amistad y los momentos vividos. A Johana, por ser una fuente de serenidad, siempre dispuesta a ayudar. A Lucía, por el apoyo las últimas semanas y su compañía en la biblioteca. A Marina, por los años que tuvimos y los conciertos que vivimos. Gracias a todos por vuestra amistad y generosidad.

A mi familia, por su confianza y apoyo incondicional. Esto incluye por supuesto a mis padres, Gloria y Pedro, que a pesar de todo y sobre todo, se han preocupado de mi educación y bien estar. Gracias por darme todo lo que tengo. Gracias también a mi hermano Gorka, por todos los momentos que hemos vivido juntos a lo largo de nuestra vida, y los que vendrán. Sé que siempre podré contar con vosotros. También con Wilco, dondequiera que esté.

Quiero agradecer especialmente a Beatriz, por su infinita paciencia, incansable apoyo y comprensión. Por recordarme cada día que los humanos nos bastamos para invocar el infierno en la tierra y llevar a quienes queremos al cielo, construirlo ahí mismo donde nos plazca. Por construir ese cielo a mi alrededor cada vez que el infierno acechaba. Por decirme: *If you are going through hell, keep going*. Por los viajes, ya eternos, que hemos hecho y que seguiremos haciendo por todo el mundo y más allá. Gracias por todo.

También, aunque sé que no podrá leer esto hasta dentro de muchos años, a ella, le quiero contar un cuento. Naiara, este año llegaste al mundo, y la noche del día en que naciste la pasé pensando que desde ese mismo momento, ya eras inmortal, precisamente porque, como bien dice Borges, no te sabes todavía mortal, pues de momento ignoras la muerte, lo divino, lo terrible, y lo incomprensible. A ti, y a tus padres Gorka y Soraya, os quiero dedicar un cuento que escribí aquella noche. Sé que ellos lo entenderán y espero que pronto te lo puedan contar.

Hace mucho tiempo, en un reino perdido... Hacía ya mucho tiempo que la Luna lloraba y lloraba y que su llanto no cesaba. Noche tras noche, lloraba, pues creía que ya no brillaba. Una noche, lloró tanto que derramando una lágrima arrastró una estrella que navegaba por las Pléyades del firmamento, esa estrella era Soraya. Y la lágrima y la estrella formaron una fuente, y al verse la Luna reflejada en ella pudo ésta ver que en efecto sí brillaba. Y ya nunca más la Luna lloraba pues siempre que dudaba miraba a Soraya y brillaba.

Un día el Sol se quejaba, estaba nublado! Y no aclaraba, no había forma de que el sol brillara. Así que el Sol, con su infinito poder, envío un rayo a la tierra, nada menos que hasta Capadocia en la lejana Turquía. Y allí surgió Gorka. Y Gorka viajó y viajó, y allí por donde fue, imperó el sol y nunca se nublaba.

Un día, viajando viajando, encontró un dragón que custodiaba una fuente. Valientemente, los ciudadanos del pueblo trataban cada día de apartar al dragón para poder beber de la fuente. Más no podían. La fuente, triste porque la luna ya no la miraba, por miedo a que el dragón la atacara, comenzaba a evaporarse.

Así que Gorka avanzó con coraje hacia el dragón, luchó, se batió con él durante largos días y largas noches. Pero a pesar de todo el esfuerzo, el dragón venció. Éste, para evitar que jamás Gorka y Soraya pudieran encontrarse, colocó a Gorka cada luna en un castillo distinto y a Soraya, la escondió en la esquina más remota del reino.

Pasaron los días, las semanas, y los años. Y ellos sin saberlo, cada noche, miraban el mismo cielo, buscándose entre las estrellas. Un día, Soraya le pidió a la Luna que por favor brillara, que brillara sobre ella para que Gorka la encontrara. Gorka vio la Luna, lejos, muy lejos, más allá de los montes, y encerrado en su torre lloraba pues él sólo jamas podría alcanzarla. Soraya sabía que jamás podría encontrarle y mirando a la Luna también lloraba. Lloraron tanto que las lágrimas de ambos formaron un río que más allá se convirtió en ría hasta dar al mar del lugar más remoto del reino.

Aquella noche la Luna llena brillaba más intensamente que nunca y todas las estrellas se reflejaban en el río, que rodeado de flores comenzó a producir un murmullo. La ría se agitaba a pesar de no haber tormenta, la mar arrullaba rompiendo el silencio, las estrellas titilaban alumbrando la inminente llegada. Rompiendo las aguas surgió la diosa de los lagos y los ríos que mediante el rocío del cielo y el agua del océano derrotó al dragón, liberando a Gorka y a Soraya de su injusto arresto.

La diosa deseada de los lagos y los ríos, de las flores y montañas, de los mares y las rías, la diosa que salvó la lágrima de la Luna y el rayo del Sol, uniéndolos por siempre jamás, no podría tener otro nombre, cómo no, que el de Naiara.

# Resumen

Las redes de comunicaciones han experimentado un incesante crecimiento desde el surgimiento del *World Wide Web* a través del cual han proliferado infinitud de servicios y aplicaciones con cada vez más usuarios interconectados. La Web e Internet han revolucionado la manera de hacer negocios de multitud de empresas, permitiendo ofrecer sus servicios de manera remota a usuarios en cualquier lugar del mundo. Igualmente, los usuarios de la red han pasado de ser actores pasivos de la red a tener un rol cada vez más activo, siendo los principales productores de contenido.

El rápido desarrollo de Internet ha producido un ecosistema de redes diversas y heterogéneas con multitud de componentes y protocolos para interconectarlos. Todo esto ha convertido el análisis de datos y la monitorización de redes en tareas cada vez más complejas. La gestión de red resulta crucial para administrar grandes centros de datos e infraestructuras de TI. La medición de parámetros de calidad de servicio (QoS) y calidad de experiencia (QoE) es esencial para determinar el estado y la calidad de los servicios ofrecidos, reduciendo y conteniendo los costes capitales y de operación.

Sin embargo, las crecientes tasas de red han abierto nuevos frentes en el área de la telemática que deberán ser abordados con nuevas metodologías que permitan manejar dichas cantidades de tráfico. Estos desafíos afectan a las etapas de captura, disección, procesamiento, análisis y visualización de los datos.

El objetivo principal de este trabajo es analizar y proponer nuevas técnicas para cada una de estas etapas, estudiando los factores clave que afectan al rendimiento de cada una de ellas. En lugar de realizar un estudio particular sobre una de estas etapas en profundidad, en este trabajo se ha optado por un enfoque horizontal que abarca múltiples etapas, lo que permite obtener una visión global del problema.

Cada una de estas etapas presenta distintas necesidades que afectan a los requisitos de versatilidad, velocidad y funcionalidad, los cuales afectan directamente al desarrollo de sus correspondientes soluciones. Por ejemplo, no se requiere de la misma versatilidad para desarrollar un sistema de alarmas que para el desarrollo de un disector de alto rendimiento para un determinado protocolo.

En este trabajo se ha propuesto una solución para la disección de tráfico HTTP capaz de manejar tráfico a tasas multi-Gbps y de distribuir la carga de trabajo equitativamente para su procesamiento en paralelo. De igual manera, se ha planteado un sistema de recolección, almacenamiento y procesamiento de mensajes de *log* apto para las nuevas necesidades de las redes heterogéneas. Igualmente, se ha propuesto un sistema de análisis a alto nivel para la generación de informes automáticos del estado de la red, desarrollando métodos para la lectura a alta velocidad de registros enriquecidos y detección de comportamientos anómalos, entre otros. Se han estudiado las implicaciones y retos de las arquitecturas desagregadas así como su impacto en tareas como el análisis masivo de datos. Por último, se ha consolidado todo lo aprendido en un proyecto de monitorización en tiempo real en una red empresarial, desarrollando técnicas para la detección de anomalías y la producción de alarmas.

Con el desarrollo de cada una de estas etapas se ha producido una serie de mejoras cuantitativas y cualitativas para los sistemas de gestión de red. En primer lugar, una herramienta de disección de tráfico HTTP capaz de soportar tasas de 11 Gbps por instancia. Seguidamente, un sistema de centralización de *logs* capaz de soportar entre 3 y 10 millones de mensajes por segundo. Posteriormente, un sistema automatizado de análisis a alto nivel capaz de generar informes proactivos del tráfico diario de forma puntual, para su uso por los administradores de la red. Finalmente, se ha desarrollado y desplegado un sistema de monitorización en un entorno real capaz de indexar 7 millones de datos diarios correspondientes a distintos componentes de la red (más de 100 VLANs, decenas de *routers* y *pooles*) y gestionar alarmas de más de 300 series temporales en tiempo real.

Para consumar los desarrollos mencionados se han identificado y resuelto distintos desafíos del ámbito de las redes de computadoras y las ciencias de la computación. Se espera que todas estas aportaciones sirvan para ampliar el conocimiento en el área de la gestión de redes y contribuyan a resolver los retos del análisis masivo de datos en redes heterogéneas.

# Abstract

Communication networks have experienced an incessant growth since the emergence of the *World Wide Web*, through which countless services and applications have proliferated, producing a diverse and heterogeneous network ecosystem with numerous components and protocols to interconnect them.

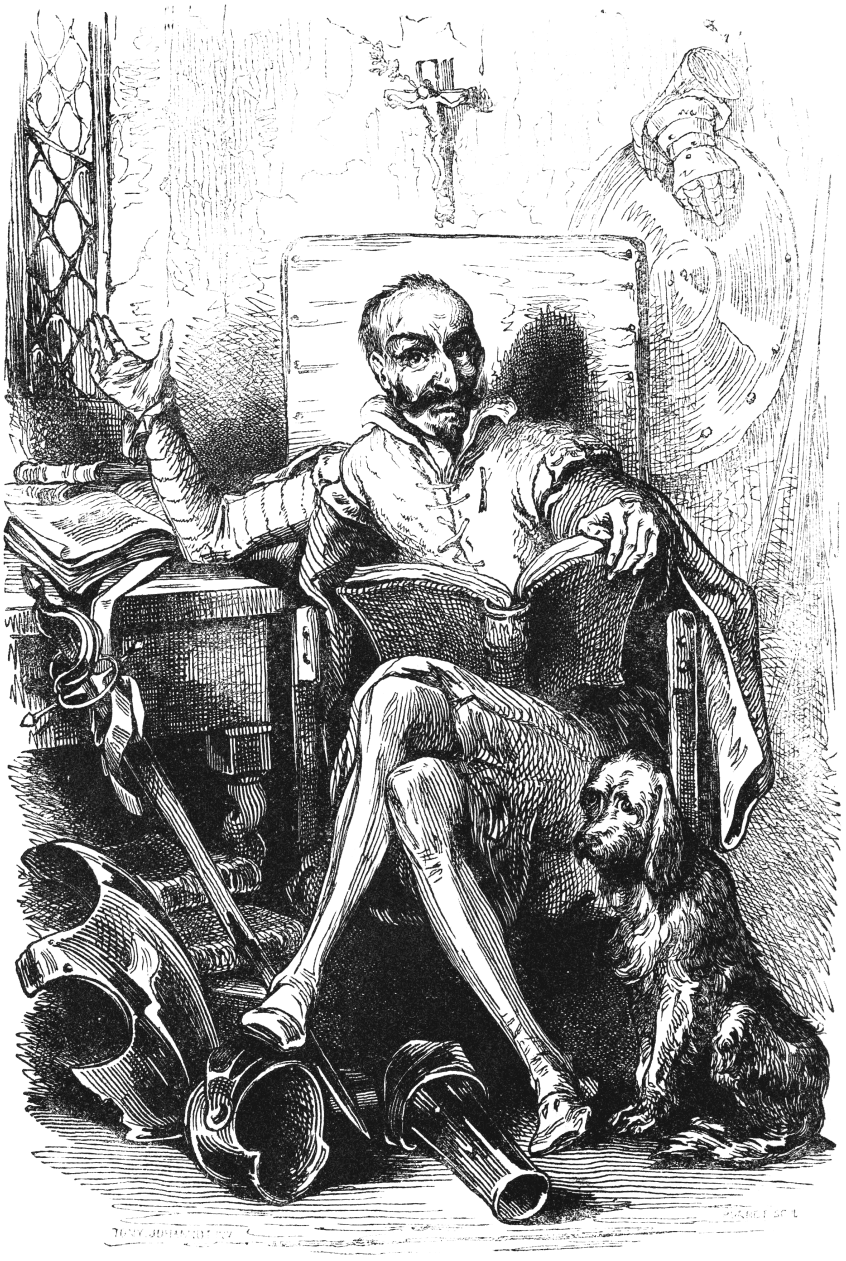
All of this has turned data analysis and network monitoring into increasingly complex tasks. Measurement of parameters such as the quality of service (QoS) or experience (QoE) in big data centers and IT infraestructures is essential to determine the status and quality of the offered services, reducing both capital expenditure (CAPEX) and operational costs (OPEX).

Notwithstanding, the growing network rates present new challenges in the telematic field, which must be addressed through new methodologies able to handle such traffic volumes. These challenges affect several stages such as the capture, dissection, process, analysis and visualization of data. Hence, the main aim of this work is to analyse and propose novel techniques for each of those stages, studying the key factors that impact their performance. This work opts for an horizontal approach that ranges multiple stages, offering a global vision on the issue. Each of the aforementioned stages presents different challenges and needs that affect the requirements of versatility, throughput and functionality, which impact the development of their corresponding solutions.

Firstly, this thesis proposes a solution for HTTP traffic dissection able to handle multi-Gb/s traffic rates and distribute the work load evenly for the sake of concurrency. In the same way, this work presents a centralized system for log collection, storing and processing suitable for the upcoming needs of the heterogeneous networks. Furthermore, a high level automated report system is proposed, along with new techniques for high performance processing of enriched records and anomaly detection, among others. In this context, the implications and challenges of dissagregated architectures have been

studied together with their impact on data analysis tasks. Lastly, all this knowledge has been consolidated for the deployment of a real-time monitoring project on an enterprise network, developing techniques for anomaly detection and alarm generation.

To achieve the aforementioned developments, many challenges regarding computer networks and computer sciences have been pinpointed and solved. We hope that the academic and industrial contributions presented in this thesis will serve to broad the knowladge in the network management area, helping to solve the many challenges of massive data analysis in heterogeneous networks.



El Ingenioso hidalgo Don Quijote de la Mancha, Barcelona,
Imprenta de Antonio Bergnes y Cía. 1839-1840.

# El Inmortal

Ser inmortal es baladí; menos el hombre, todas las criaturas lo son, pues ignoran la muerte; lo divino, lo terrible, lo incomprensible, es saberse inmortal.

*[...]*

El pensamiento más fugaz obedece a un dibujo invisible y puede coronar, o inaugurar, una forma secreta. Sé de quienes obraban el mal para que en los siglos futuros resultara el bien, o hubiera resultado en los ya pretéritos. Encarados así, todos nuestros actos son justos, pero también son indiferentes. No hay méritos morales o intelectuales. Homero compuso la Odisea; postulado un plazo infinito, con infinitas circunstancias y cambios, lo imposible es no componer, siquiera una vez, la Odisea. Nadie es alguien, un solo hombre inmortal es todos los hombres. Como Cornelio Agrippa, soy dios, soy héroe, soy filósofo, soy demonio y soy mundo, lo cual es una fatigosa manera de decir que no soy.

*[...]*

Cabe en estas palabras: Existe un río cuyas aguas dan la inmortalidad; en alguna región habrá otro río cuyas aguas la borren. El número de ríos no es infinito; un viajero inmortal que recorra el mundo acabará, algún día, por haber bebido de todos. Nos propusimos descubrir ese río.

*[...]*

La muerte (o su alusión) hace preciosos y patéticos a los hombres. Éstos se conmueven por su condición de fantasmas; cada acto que ejecutan puede ser el último; no hay rostro que no esté por desdibujarse como el rostro de un sueño. Todo, entre los mortales, tiene el valor de lo irrecuperable y de lo azaroso. Entre los Inmortales, en cambio, cada acto (y cada pensamiento) es el eco de otros que en el pasado lo antecedieron, sin principio visible, o el fiel presagio de otros que en el futuro lo repetirán hasta el vértigo. No hay cosa que no esté como perdida entre infatigables espejos. Nada puede ocurrir una sola vez, nada es preciosamente precario. Lo elegíaco, lo grave, lo ceremonial, no rigen para los Inmortales.

*Jorge Luis Borges*

# Tabla de Contenidos

# ÍNDICE DE CUADROS

# ÍNDICE DE FIGURAS

**FIGURA** **Página**

# Notas de código

**Página**

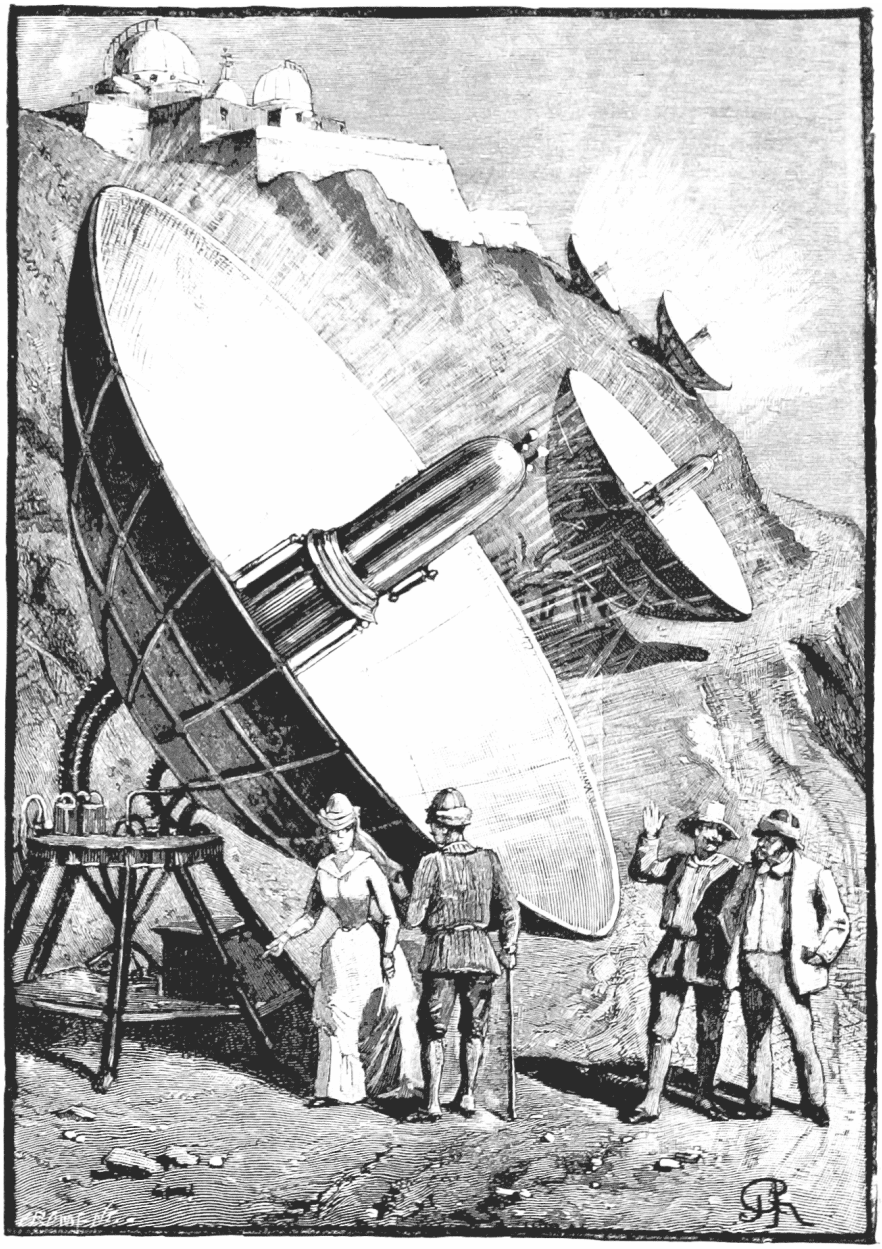


Ilustración del libro *Les Exilés de la Terre* de André Laurie (pseud. de Paschal Grousset) traducido al inglés como *The Conquest Of The Moon: A Story of the Bayouda*. 1889

CAPÍTULO 1

# INTRODUCCIÓN

La era de la información ha transformado la manera en que las sociedades se comunican a un nivel sólo comparable con la revolución producida tras el desarrollo de la imprenta moderna con tipos móviles de Johannes Gutenberg, permitiendo la transmisión de información a cualquier lugar del mundo instantáneamente mediante la interconexión de los ordenadores, formando la red de redes.

Las primeras aplicaciones de Internet (e-mail, BBS) permitieron a sus usuarios establecer discusiones (Usenet), intercambiar ficheros (FTP), y tan pronto su adopción comenzó a crecer, entró también en juego el comercio electrónico.

Precursores como Ibertex [1] o Minitel permitieron a millones de personas realizar compras y consultas a través de la línea telefónica, pero fue la llegada del *World Wide Web* lo que finalmente cambió la manera en que las personas realizaban tanto sus tareas más cotidianas, como la compra de alimentos, hasta las más trascendentales, como la forma de informarse o socializar.

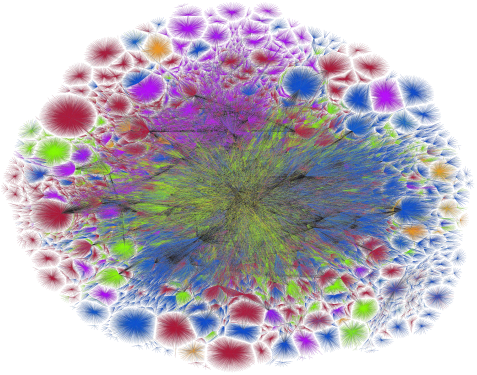



Figura 1.1
Internet en 2015. Imagen de Opte Project

La globalización de Internet ocurrió a partir del momento en que el valor de la red compensaba el coste de pertenecer a ella. La ley Metcalfe [2] ha sido usada para explicar este fenómeno en el crecimiento de diversas tecnologías como la telefonía móvil, las aplicaciones web o las redes sociales. Intuitivamente, a medida que crece el número de nodos (individuos o dispositivos) de una red, se incrementa su interconectividad (ver Fig. 1.2), y si los nodos pueden enlazar al contenido de otros nodos, el valor de la red aumenta rápidamente [3].

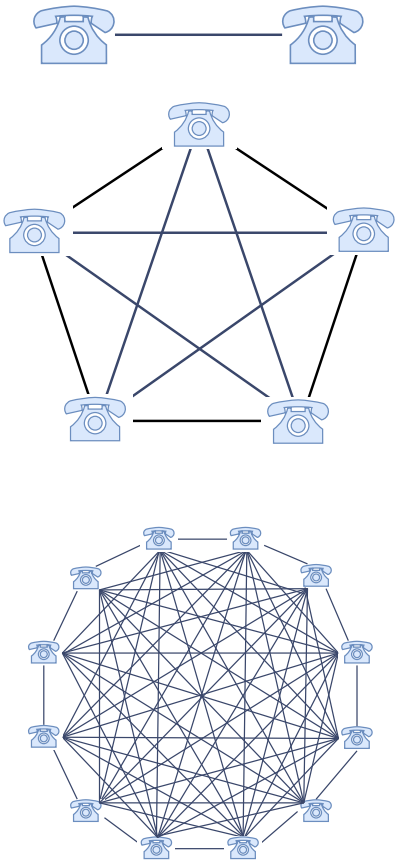


Figura 1.2
Dos teléfonos pueden establecer tan sólo una conexión, 5 pueden hacer 10 conexiones, y 12 pueden realizar 66.

La Web e Internet generaron un mar de oportunidades para comercios y empresas, permitiendo el acceso a un sinfín de servicios de manera remota y económica. Posteriormente, la Web 2.0 permitió a los usuarios no sólo consultar contenido sino interactuar y colaborar entre sí. A partir de este momento los usuarios no sólo consumen contenidos sino que también los producen.

La gran cantidad de datos producida en la Web, junto con la popularización de los teléfonos inteligentes y la mejora de las redes móviles (3G, LTE) han impulsado el desarrollo de la computación en la nube, permitiendo el acceso ubicuo a la información tanto a empresas como a usuarios. Más recientemente, el Internet de las cosas amenaza con interconectar millones de dispositivos (*wearables*, sensores, etc.), permitiendo el desarrollo de casas y ciudades inteligentes, entre otros.

Este vertiginoso desarrollo de Internet junto con el incesante crecimiento del número de usuarios y dispositivos conectados a la red, genera numerosos retos técnicos en el ámbito de la telemática. La amplitud de estos problemas abarca desde la propia arquitectura de la red de computadoras hasta el análisis y monitorización de las redes, pasando por el procesamiento y almacenamiento masivo de datos.

En este trabajo, se explorarán las distintas etapas involucradas en el proceso monitorización del tráfico, como son la recolección, el análisis y la visualización del tráfico en las redes de computadoras; realizando contribuciones en cada uno de estos procesos.

## 1.1. Motivación

El continuo aumento en la cantidad de datos transmitidos por la red se ha experimentado en todos los niveles que forman la red de redes, y se observa tanto en las columnas vertebrales de Internet como en las redes domésticas. Tal como se muestra en la tabla 1.1, en los próximos años se espera un incremento del tráfico general, con un incremento de más de 25 Exabytes mensuales [4] entre el año 2016 y 2017 hasta alcanzar los 121 Exabytes mensuales en 2017, llegando a los 278 Exabytes mensuales en el año 2021.

Este crecimiento repercute en el tamaño de los grandes centros de datos [82] y del volumen de datos [39, 40] que gestionan, requiriendo nuevas técnicas de captura de tráfico adaptadas a redes multi-Gb/s de 10, 40 y 100 Gbps [85, 86], que permitan realizar auditorías para una adecuada gestión de las redes.

A grandes rasgos, el proceso de monitorización de redes y los componentes presentes en ellas requiere primero la extracción de información, tanto mediante la captura del tráfico como con la recolección de mensajes de *log* de las distintas aplicaciones presentes en los nodos de la red. Posteriormente, la información recolectada deberá ser filtrada, agregada y/o transformada en función de los requisitos del análisis a realizar, concentrando la información relevante y descartando la información fútil. Los sistemas de análisis deberán ser versátiles e inteligentes, permitiendo una adaptación ágil ante nuevos problemas, para su empleo proactivo. Finalmente, la visualización de los datos debe ser clara y concisa, permitiendo una rápida identificación de las incidencias para proporcionar una respuesta reactiva.

Cuadro 1.1: Crecimiento del tráfico IP global

Datos del Cisco Visual Networking Index: Forecast and Methodology 2016–2021

| | Por Tipo Petabytes (PB) por mes | | | Por Segmento PB/mes | | Total PB/mes |
|---|---|---|---|---|---|---|
| | Internet Fijo | IPs Gestionadas[1] | Datos Móviles | Consumidores | Negocios | Tráfico IP Total |
| 2016 | 65.942 | 22.911 | 7.201 | 78.250 | 17.804 | 96.054 |
| 2017 | 83.371 | 27.140 | 11.183 | 99.777 | 21.917 | 121.694 |
| 2018 | 102.960 | 31.304 | 16.646 | 124.689 | 26.220 | 150.910 |
| 2019 | 127.008 | 35.226 | 24.220 | 154.935 | 31.518 | 186.453 |
| 2020 | 155.121 | 38.908 | 34.382 | 190.474 | 37.937 | 228.411 |
| 2021 | 187.386 | 42.452 | 48.270 | 232.655 | 45.452 | 278.108 |

[1] Incluye tráfico WAN corporativo, el transporte IP de TV y Vídeo bajo demanda.

El objetivo principal de este trabajo es analizar y proponer nuevas metodologías para distintos casos de uso en cada una de las etapas mencionadas con el fin de adaptar las técnicas a redes cada vez más heterogéneas, con una proliferación mayor de sistemas y aplicaciones, y un incesante aumento del tráfico. Para ello, a lo largo de este trabajo se revisarán los distintos conceptos fundamentales de cada fase, junto con un estado del arte con las diversas metodologías más relevantes del área estudiada, atendiendo a los diferentes tipos de requisitos demandados por cada tarea, y proponiendo soluciones para los distintos retos encontrados.

Antes de continuar, se ha de recalcar el carácter pragmático e industrial de este trabajo, cuyas aportaciones convergen hacia la resolución de problemas concretos en el entorno de la telemática y la ingeniería. Es importante destacar que paralelamente a la producción académica en forma de distintas publicaciones elaboradas a lo largo de estos años, el trabajo realizado también ha facilitado la generación de una transferencia industrial, la cual ha repercutido directamente en el desarrollo de varios proyectos reales y la resolución de problemas concretos en entornos empresariales.

La monitorización en grandes centros de datos resulta vital para mantener un alto grado de *Calidad de Servicio*, la cual está fuertemente ligada con la *Calidad de Experiencia* [8] de los usuarios de los distintos servicios ofrecidos. Una correcta monitorización de las infraestructuras de red y los servicios permite prevenir problemas y localizar el origen de las incidencias, reduciendo los costes operacionales de las infraestructuras. Por otro lado resulta esencial en el estudio de la capacidad de los componentes de la red, permitiendo realizar evaluaciones para un apropiado escalado de los recursos, ampliando los componentes presentes en el centro de datos en función de los cambios en los servicios ofrecidos.

## 1.2. Retos y estructura del trabajo realizado

Durante la realización del presente trabajo se han identificado distintos retos en cada una de las etapas anteriormente mencioandas (ver figura 1.3). En primer lugar, los trabajos de mejora de los controladores (*drivers*) de tarjetas de red, como HPCAP [31] para redes 10GbE, siguen adelante con el fin de adaptar el proceso de captura de datos a las crecientes tasas de red [5] de 20 y 40 Gbps. A pesar de que la mejora de estos sistemas no entra en el ámbito de este trabajo, su vinculación inherente con el resto de etapas de la cadena de monitorización requiere al menos de su mención y consideración en el análisis de requisitos del resto de etapas a contemplar.

Durante el estudio de las etapas de recolección y disección de información se han

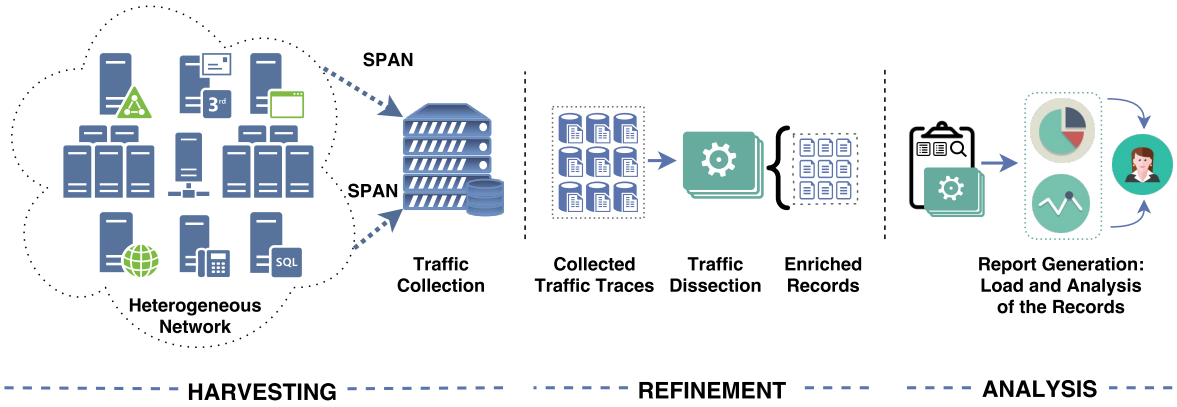



Figura 1.3: Distintas etapas del proceso de monitorización de red.

reconocido los distintos inconvenientes a la hora de distribuir la carga de trabajo de esta disección entre varios procesos concurrentes. Los detalles de este estudio se abordan en profundidad en el capítulo 2º, *HTTPanalyzer*: Disección del Tráfico HTTP del presente trabajo. Entre las contribuciones se cuenta con el desarrollo de una herramienta para la disección del tráfico HTTP a alta velocidad, estudiando los factores que permiten realizar esta tarea a velocidades multi-Gb/s y las limitaciones encontradas. Adicionalmente, se propone una metodología de balanceo de carga para el caso particular del procesado de tráfico HTTP para hacer frente a altas tasas mediante múltiples instancias de HTTPanalyzer.

Posteriormente, en el capítulo 3º, *Loginson*: Recolección y Análisis de Registros, se expone un sistema de centralización de alto rendimiento para la recolección y procesamiento de mensajes de *log* a gran escala, manteniendo una proporcionalidad en cuanto a los recursos empleados para tal menester, y estudiando las distintas alternativas disponibles en el estado del arte. Este punto sobre la proporcionalidad de los recursos será mencionado recurrentemente a lo largo de los próximos capítulos debido a su importancia a la hora de dimensionar un sistema de monitorización de acuerdo al tamaño del sistema monitorizado. El desafío en este caso consiste en conseguir un rendimiento por encima del estado del arte del momento de realización de esta investigación, mediante unos recursos acotados, con una alta escalabilidad vertical y adaptabilidad a distintos tipos de *logs*, proveyendo a su vez una vista de alto nivel para la monitorización reactiva de los datos.

En la misma línea, en lo correspondiente a la etapa de análisis, se ha identificado la necesidad de mejorar los sistemas de análisis proactivo de grandes infraestructuras de TI. Los detalles de este estudio y sus contribuciones se detallan en el capítu-

lo 4º, *FERMIN*: Automatizando el Análisis del Tráfico. Las aportaciones de este estudio se centran en la mejora de la versatilidad del análisis mediante el uso de técnicas de alto nivel, y a su vez en la integración de distintas técnicas de alto rendimiento para la lectura y análisis a alta velocidad de grandes cantidades de datos. Este trabajo contribuye en el ámbito de la auditoría de redes con el desarrollo de una solución para el análisis del tráfico, mediante la generación automática de informes, con el fin de mejorar el análisis proactivo en grandes organizaciones.

En el marco del análisis de datos, se han estudiado los distintos desafíos de las arquitecturas desagregadas, y la propuesta del proyecto europeo *Disaggregated Recursive Datacentre-in-a-Box* para mejorar la proporcionalidad en el uso de recursos de los grandes centros de datos, así como la viabilidad de este tipo de arquitecturas para tareas intensivas como el análisis de datos para el análisis del estado de redes de computadoras. A esta cuestión está dedicado el capítulo 5º, Diluyendo las Barreras de la Escalabilidad.

Finalmente, todo este conocimiento se ha consolidado con su aplicación en distintos proyectos reales, mediante el despliegue de sistemas de monitorización, alarmas y auditoría de redes en infraestructuras heterogéneas, con los cuales se ha obteniendo una valiosa experiencia y retroalimentado la investigación realizada.

En concreto, en el capítulo 6º, Monitorización de redes, se describen los detalles del despliegue de un sistema de monitorización en la *Sociedad Estatal de Correos y Telégrafos* de España, como parte de un proyecto de *Naudit*. De nuevo, juegan un papel esencial los conceptos de proporcionalidad, versatilidad y alto rendimiento a la hora de enfrentar los distintos obstáculos. En el capítulo mencionado se explican una serie de metodologías simples pero potentes aplicadas a la monitorización de la infraestructura de la *Sociedad Estatal de Correos y Telégrafos*.

## 1.3. Contexto y trabajo previo

Previamente al inicio del doctorado, se realizaron trabajos en el laboratorio High Performance Computing and Networking (HPCN) orientados al análisis de tráfico y la mejora de los sistemas de visualización de datos haciendo uso de técnicas Web. Como parte del trabajo de fin de máster [6] se realizó un diseño y primer desarrollo de una herramienta de disección de tráfico HTTP, comparándola con otras herramientas del grupo de investigación. Este trabajo se continuaría hasta lo que hoy es *HTTPanalyzer* mejorando su velocidad y características, complementándolo con una comparativa con otras herramientas de renombre.

Durante los primeros años en el grupo HPCN también se exploraron las distintas técnicas de visualización de datos, y las distintas librerías para desarrollo web de esta clase de soluciones orientadas a cantidades masivas de datos. Se realizaron pequeños programas de visualización para topologías, generación de gráficas estáticas a partir de grandes conjuntos de datos, generación de gráficos interactivos mediante librerías JavaScript, etcétera. Estas técnicas han ido mejorando a lo largo de los años, ofreciendo mejoras para la visualización Web y generación de gráficos. Estos sistemas recientes se han empleado en los desarrollos expuestos en este trabajo.

Al principio del doctorado, buscando consolidar la línea de investigación, se realizaron varios trabajos. Entre ellos destaca el repositorio de tráfico distribuido *EINSrepo* para el proyecto Network of Excellence in Internet Science (EINS) en colaboración con el *Nexa Center for Internet & Society*[1] de la universidad Politecnico di Torino. Mediante este proyecto pudo estudiarse la viabilidad de los sistemas de Hadoop y similares para tareas de esta índole. A partir de este trabajo se adquirió la motivación para evaluar otros sistemas similares, lo que resultaría en el extenso estado del arte realizado para el proyecto *Loginson*, descrito en el capítulo 3º, *Loginson*: Recolección y Análisis de Registros. En 2015, el proyecto EINSrepo fue presentado en forma de póster en la 2ª Conferencia Internacional sobre Ciencias de Internet en Bruselas, Bélgica.

La unión de estos trabajos motivó la realización del presente trabajo definiendo una línea de investigación clara, en la cual se estudian las distintas etapas que conforman el proceso de monitorización de redes y los distintos retos desde el punto de vista de la computación de alto rendimiento. Las aportaciones realizadas se enmarcan en el ámbito del análisis de datos, las técnicas de alto rendimiento para la adaptación de estas metodologías a nuevas tasas de red y los métodos de visualización de datos y su capacidad para manejar grandes conjuntos de datos.

[1] `https://nexa.polito.it/`

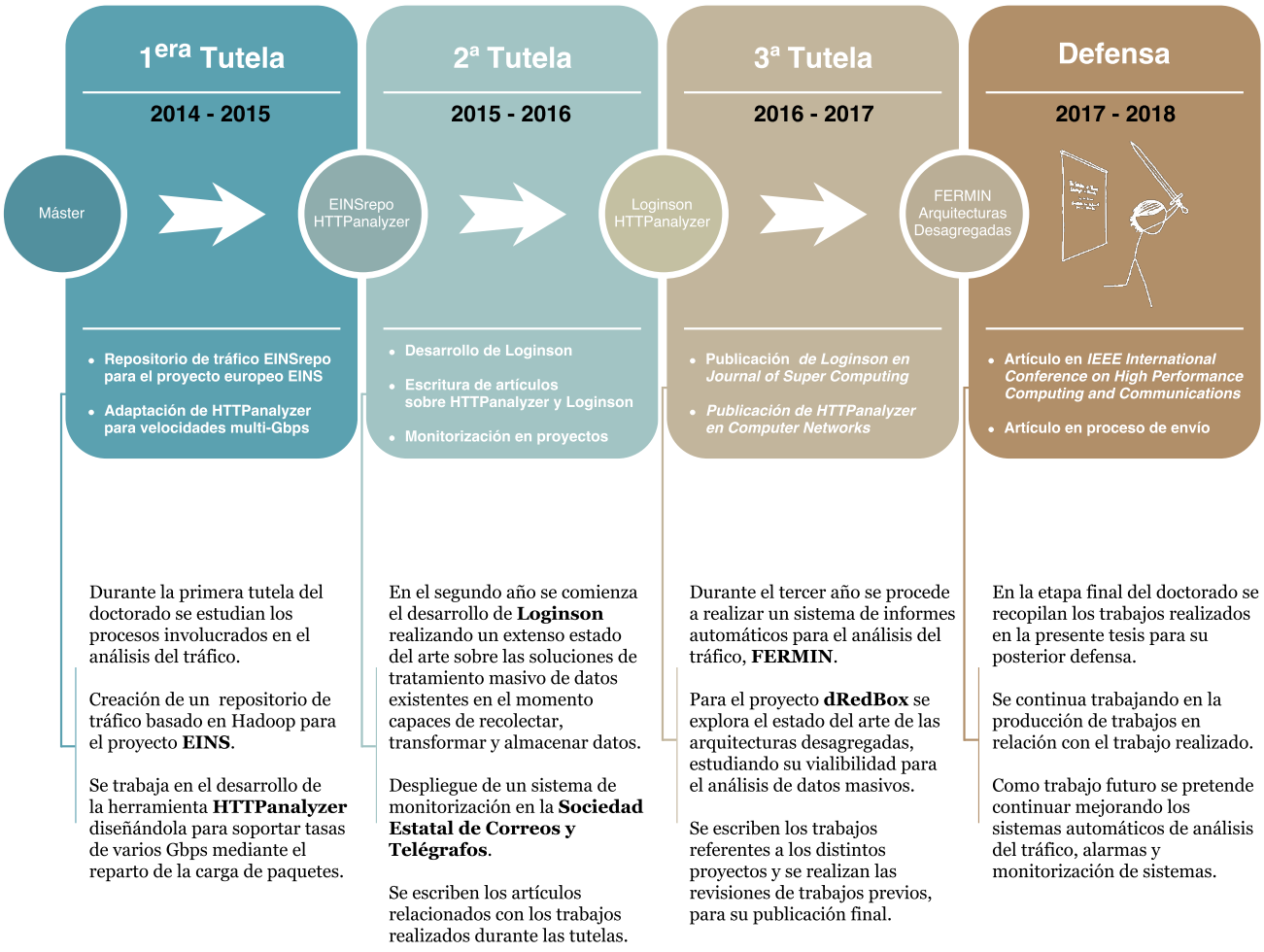



Figura 1.4: Simplificación del proceso de realización del presente trabajo.

Golden Harvest. 1894

CAPÍTULO 2

# *HTTPanalyzer*: DISECCIÓN DEL TRÁFICO HTTP

En este estudio se propuso e implementó una serie de técnicas originales para el análisis del tráfico web Hypertext Transfer Protocol (HTTP) y sus métricas asociadas (tiempos de respuesta, códigos de respuesta, etcétera), sin necesidad de realizar un reensamblado de la conexión TCP subyacente, lo cual perjudicaría gravemente el rendimiento del análisis de tráfico debido al proceso adicional que requeriría. Además, la técnica propuesta para el análisis del tráfico HTTP se puede ejecutar en hardware convencional, permitiendo un despliegue barato del sistema en sistemas convencionales.

Con este fin, se presenta una técnica de balanceo de carga a nivel de conexión sub-TCP que incrementa significativamente el rendimiento del análisis a costa de cierta pérdida de precisión, por ejemplo, una ligera pérdida en los datos de transacciones HTTP. Estas técnicas proveen estadísticos para la evaluación del rendimiento que son indistinguibles de los resultados ofrecidos por las alternativas mono-hilo con reensamblado completo de las conexiones TCP. Adicionalmente, se propone una técnica para el balanceo de carga del tráfico HTTP que permite su procesado en paralelo soportando altas tasas multi-Gbps.

## 2.1. Introducción al análisis del tráfico HTTP

Los negocios y operaciones de las grandes organizaciones, como por ejemplo bancos y corporaciones industriales, se realizan cada vez más a través de Internet [39], el cual, durante los últimos años, se ha convertido en un medio esencial a través del que ofrecer sus servicios. Usualmente, HTTP es el protocolo de elección para proveer estos servicios al usuario final, en mayor medida gracias a la gran adopción de los clientes web en todo tipo de dispositivos móviles o de escritorio, y a la facilidad que ofrece para diseñar Interfaces de Programación de Aplicaciones (*APIs*) acordes con el diseño de Transferencia de Estado Representacional (*REST*). Por tanto, la medición de la calidad de servicio otorgada por los portales web [8] tiene una importancia estratégica fundamental para los proveedores de servicios. Esto mismo es válido para otros muchos protocolos de nivel de aplicación (VoIP, SIP, RTP, RTCP) [9], pero el estudio en esta etapa de la investigación se ha centrado en el tráfico HTTP, debido principalmente a su gran adopción y presencia.

Esta evaluación de la calidad de servicio está normalmente basada en estadísticas derivadas del tiempo de respuesta, desde la petición HTTP hasta su correspondiente respuesta, así como en el análisis de los códigos de respuesta para la detección de comportamientos anómalos mediante los sistemas de monitorización de los servicios web. Por ejemplo, un error HTTP con código 500 indica la ocurrencia de un error interno en el servidor, y por tanto, deberá ser tomado en cuenta si la cantidad de estos errores alcanza una cifra significativa.

La disección y el análisis del tráfico HTTP puede ser realizada también por motivos de ciberseguridad, sin embargo, este tipo de análisis requiere un nivel de detalle mucho mayor al de las estadísticas agregadas, debido a que las amenazas de seguridad pueden tratar de ocultarse entre el tráfico HTTP benigno. Por tanto, evitar la pérdida de las transacciones HTTP es crucial y fundamental para esta clase de análisis forense y de seguridad. Por otro lado, el ámbito de la investigación realizada en esta etapa se centra, no en el análisis forense, sino en la monitorización de los servicios de red, en los cuales, sólo son relevantes las estadísticas agregadas como las medias, promedios, o distribuciones de probabilidad, entre otros.

Por supuesto, para la evaluación de la calidad de servicio, sólo se requieren estadísticos agregados. A saber: tiempo de respuesta global o el porcentaje de ocurrencia de cierto tipo de códigos de error. Además, estos estadísticos deberán ser ofrecidos en tiempo real con el propósito de reaccionar lo más rápidamente a imprevistos y anomalías. Una vez que estos estadísticos agregados muestren una degradación del rendimiento, un análisis

en profundidad deberá ser realizado mediante la inspección en diferido de los paquetes que componen el tráfico afectado en el intervalo de tiempo concreto de la incidencia.

Teniendo en cuenta lo anterior, la herramienta de análisis del tráfico HTTP debe ser lo suficientemente ágil para hacer frente a tasas de varios Gbit (multi-Gb/s) por segundo y proveer estadísticos agregados en tiempo real, en vez de ofrecer una muy alta precisión a expensas de un gran tiempo de procesamiento. En este trabajo se propuso:

- 1) Aligerar el reensamblado de la conexión TCP subyacente y hacer uso de una técnica novedosa para el balanceo de carga con el fin de soportar grandes cantidades de tráfico manteniendo a su vez la precisión a un nivel razonable.
- 2) Mediante lo anterior, proveer estadísticas agregadas en tiempo real a partir del tráfico HTTP, tales como tiempos de respuesta y códigos de respuesta, entre otros. Además, también se alcanzó un procesamiento sostenido de 20 Gbps (2 líneas de 10 Gbps) en una única máquina mediante la utilización de varias instancias del disector de tráfico ejecutándose en paralelo.

Las técnicas propuestas han sido implementadas en la herramienta *HTTPanalyzer* como prueba de concepto y banco de pruebas para la evaluación de su rendimiento. Dos escenarios con tráfico de real provenientes de grandes portales comerciales han sido utilizados para la evaluación del rendimiento máximo ofrecido por la herramienta así como de la evaluación de la precisión de sus estadísticas de calidad de servicio.

El resto del capítulo está organizado de la siguiente manera: Primero, antes de proceder con el contenido técnico, se realiza una revisión del estado del arte en esta materia. Segundo, se describe la metodología utilizada y las técnicas propuestas para el análisis del tráfico web, las cuales están basadas en el conocimiento parcial de las conexiones TCP y el balanceo de la carga a analizar mediante la información a nivel la conexión sub-TCP. Finalmente, se discuten los resultados del rendimiento ofrecido por la herramienta así como de la precisión de la información que otorga, seguido de unas conclusiones generales.

## 2.2. Estado del arte del análisis de tráfico HTTP

La mayoría de las herramientas de análisis HTTP disponibles en el estado del arte están más centradas en la fiabilidad que en el rendimiento ofrecido. En realidad, algunas de ellas realizan un análisis en diferido en el cual la velocidad de procesamiento (y por tanto el rendimiento) no resulta una prioridad en absoluto. Por tanto, son muy adecuadas

para tareas tales como el análisis forense o la evaluación de la calidad de servicio en diferido; pero no para la realización de un análisis en tiempo real a tasas de varios Gbit por segundo. Normalmente, estas herramientas están basadas en la **reconstrucción de las conexiones TCP** y, en paso subsequente, la correlación de las peticiones HTTP con sus correspondientes respuestas con el fin de obtener el tiempo de respuesta y las estadísticas del número de errores. Mientras que este procedimiento ofrece muy buenos resultados en términos de precisión, por otro lado añade un hándicap considerable que imposibilita el procesamiento de los datos a altas velocidades.

Es necesario hacer hincapié en que el protocolo HTTP 1.1 es persistente, en concreto, varias peticiones y respuestas HTTP podrían ser transmitidas a través de la misma conexión TCP. Estas peticiones y respuestas son posteriormente segmentadas en trozos y encapsuladas dentro de segmentos TCP. Gracias a los números de secuencia de cada segmento, el receptor es capaz de ordenar los segmentos recibidos fuera de orden y de eliminar los duplicados [10]. Entonces, TCP reensambla estos fragmentos creando un flujo de datos que alimenta la capa de aplicación de forma apropiada a lo que esta espera. Por tanto, el reensamblado de la conexión TCP es un primer paso obligatorio para recuperar cada una de las peticiones y respuestas HTTP en el receptor.

Debido al complejo procesamiento de reensamblado, estas herramientas del estado del arte hacen uso de sofisticados sistemas many-core y multi-core con el fin de conseguir altas velocidades de procesamiento. Por ejemplo, Jing Xu et al. [11] proponen una solución para la disección del tráfico HTTP utilizando la plataforma many-core Tilera para el análisis de tráfico HTTP en tiempo real, consiguiendo velocidades de hasta 2 Gbps. Esta solución realiza un proceso de desfragmentación IP y reensamblado TCP. Teniendo en cuenta que estos resultados son bastante buenos, es necesario apuntar que esta solución requiere de una tarjeta PCI especializada para descargar la CPU de la máquina hospedante. En este caso se trata de una TILEPro64 con 64 núcleos de procesamiento. *HTTPanalyzer* en cambio, es una solución portable y rentable que se puede utilizar en hardware convencional, y que a expensas de una menor precisión ofrece estadísticas de gran valor en tiempo real, para la mayoría de las tareas de monitorización, ya sean en línea o en diferido.

Otro ejemplo interesante es el propuesto por Zhang et al. [12] para *hardware* de propósito general, en arquitecturas multi-core de Intel, mediante la construcción de una línea de procesamiento siguiendo un modelo RTC (real-time clock), el cual, al igual que el ejemplo anterior, también reensambla las conexiones TCP. Este sistema consigue una velocidad de casi 20 Gbps cuando analiza tráfico en busca de peticiones y respuestas

HTTP mediante 5 cores. Según sus resultados, en una traza con aproximadamente 2 millones y medio de paquetes y una media de 764 bytes de longitud por paquete, la velocidad de procesamiento se mantiene en unos 3.3 millones de paquetes por segundo (Mpps). Sin embargo, al contrario que lo propuesto en *HTTPanalyzer*, las peticiones y respuestas detectadas no son emparejadas y por tanto no se obtienen estadísticas a partir de ellas, como por ejemplo, el tiempo de respuesta.

Otras herramientas como Bro, Fiddler, FlowScan, que no requieren de hardware específico, también proveen estadísticas y métricas de gran precisión a costa del rendimiento, ya que todas ellas reconstruyen la conexión TCP. Bro [13] [14] es un monitor orientado a la seguridad de la red que aplica secuencias de código y políticas a los eventos inducidos a partir de los flujos de paquetes, creando una serie de registros y alertas sobre el estado de la red analizada. Fiddler [15] es una herramienta para la depuración de los servidores HTTP orientada a plataformas Windows, y ayuda en el mantenimiento de los sistemas web analizando el tráfico entre el servidor y el navegador.

Otra herramienta de depuración a considerar es HTTPperf [16], la cual mide activamente el rendimiento de los servicios web generando tráfico HTTP con el fin de probar entornos de pre-producción. FlowScan es un paquete de software que analiza datos en formato NetFlow [17] y a partir de ello provee una representación gráfica de distintas métricas. Sin embargo, esta herramienta puede verse superada por el alto número de flujos a ser analizados, tal como se detalla:*"might not be able to scale beyond monitoring a couple fully-utilized OC3 (155 Mb/s) links."* [18] (p.314). Tener en cuenta la conexión TCP requiere un procesamiento más complejo y por tanto lento, ya que mantener en memoria el estado de miles de conexiones diferentes requiere un gran poder de procesamiento y numerosos recursos. Este es el enfoque general visto en diferentes herramientas de análisis del estado del arte.

Para el desarrollo de herramientas que reconstruyan toda la comunicación TCP y a continuación extraigan información de protocolos de aplicación (tales como HTTP), *Libnids*, by Rafal Wojtczuk [19] es una librería, ahora descontinuada, que provee funcionalidades de reensamblado del flujo de datos TCP y de desfragmentación IP así como herramientas para el escaneo de puertos y la realización de análisis en profundidad de los datos de aplicación de la capa TCP.

Se puede concluir que el estado del arte muestra que la obtención de estadísticas de alta precisión a altas velocidades puede obtenerse únicamente a través de hardware especializado. En este capítulo se detalla una solución que antepone la velocidad de procesamiento para la obtención de estadísticos agregados en hardware convencional

Cuadro 2.1: Ficheros de tráfico.

| Fichero de captura | Tamaño | Paquetes en el fichero | Transacciones HTTP |
|---|---|---|---|
| trace1.pcap | 387 GB | 539.178.347 | 13.743.811 |
| trace2.pcap | 120 GB | 211.823.223 | 3.681.812 |

facilitando su mantenimiento y despliegue para el análisis de tráfico HTTP en linea o en diferido. En concreto, la novedad expuesta se compone de dos partes. Primero, se propone una nueva herramienta de análisis HTTP con un rendimiento multi-Gbps por encima del establecido en el estado del arte, prescindiendo de la reconstrucción de los flujos TCP. Segundo, se presenta una técnica novedosa para distribuir el tráfico HTTP a nivel de transacción HTTP hacia multiples consumidores, lo cual incrementa el rendimiento.

## 2.3. Metodologías aplicadas al análisis y disección del tráfico HTTP

Para realizar los experimentos de esta etapa de la investigación se hizo uso de muestras de tráfico reales, las cuales consisten en archivos PCAP (Packet Capture) formados por tráfico HTTP obtenido en *proxies* de producción en dos redes distintas de grandes corporaciones, conteniendo estos ficheros millones de transacciones HTTP. La descripción de estos ficheros puede encontrarse en la tabla 2.1. Se escogieron dos compañías diferentes con el fin de tener una muestra larga y diversa de este tipo de tráfico. Estos ficheros fueron usados para evaluar la precisión y el rendimiento de la herramienta *HTTPanalyzer*.

En lo que respecta a la evaluación de la precisión, **Tshark** [20] [21] fue utilizado como referencia, pues hoy en día es la herramienta estándar *de facto* para el análisis de tráfico. Esta herramienta reconstruye los flujos de la capa TCP y utiliza numerosos y complejos disectores de paquetes para los distintos protocolos disponibles, ofreciendo un gran nivel de detalle a partir de las trazas de tráfico aunque siempre a costa de un procesamiento lento.

Es necesario destacar que Tshark es incapaz de procesar muestras de tráfico como las mencionadas anteriormente (ver Tabla 2.1) debido a sus grandes requisitos de memoria, los cuales son proporcionales al tamaño del fichero a analizar. Consecuentemente, las muestras de tráfico utilizadas se dividieron en trozos de aproximadamente 20 millones de paquetes, lo cual resulta en 27 trozos para el caso de la traza **trace1.pcap** y 11 trozos para el caso de la traza **trace2.pcap**; con un tamaño de fichero entre 13 y 15 GB para

cada trozo. Puesto que algunas transacciones pueden quedar divididas entre fichero y fichero, y con el fin de realizar una comparación justa, las dos listas de archivos obtenidas a partir de las muestras de tráfico fueron utilizadas para la evaluación de ambas herramientas.

En lo que se refiere a la evaluación del rendimiento, se consideraron dos escenarios diferentes, para la evaluación de la precisión y la velocidad, respectivamente. El primer caso (en diferido, ver Figura 2.1) consiste en un procesamiento en diferido de los ficheros utilizando una única instancia de *HTTPanalyzer*, con el objetivo de comparar los resultados dados por *HTTPanalyzer* contra los de Tshark. En el segundo escenario (en línea, ver Figura 2.2), se emplearon varias instancias de *HTTPanalyzer* con una nueva técnica de balanceo de carga a nivel de transacción HTTP, en vez de a nivel del reparto tradicional a nivel de flujo TCP, enfocado al procesamiento a altas velocidades (20 Gbps). Con este objetivo, se empleó una herramienta software balanceadora de tráfico llamada *PacketFeeder*, la cual recibe los paquetes desde la interfaz de red y a continuación los distribuye de forma uniforme entre los distintos consumidores (instancias de *HTTPanalyzer*) mediante memoria compartida; todo ello preservando la consistencia de las transacciones HTTP gracias a las funciones resumen (de ahora en adelante, *funciones hash*) que se explicarán más adelante en la sección 2.3.1.4. En decir, las respuestas HTTP y sus correspondientes peticiones asociadas serán enviadas desde el balanceador al mismo consumidor.

En la sección 2.4 se discutirá más detenidamente el compromiso entre precisión y velocidad de procesamiento de *HTTPanalyzer* versus Tshark.

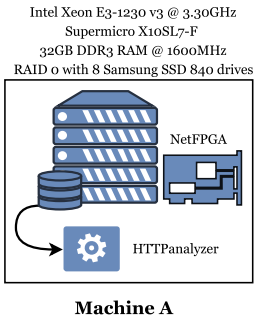



Figura 2.1: Escenario A

Evaluación en diferido con un único consumidor.

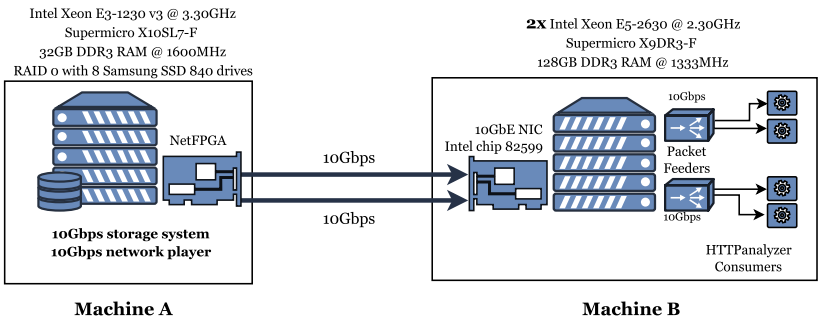



Figura 2.2: Escenario B

Evaluación en línea haciendo uso de múltiples consumidores.

Diferentes escenarios para las pruebas en diferido y en línea.

### 2.3.1. Módulos del sistema implementado

La herramienta *HTTPanalyzer* está estructurada internamente en varios módulos, en concreto: una matriz asociativa (de ahora en adelante, *tabla hash*) para las peticiones y respuestas HTTP; dos piscinas de estructuras (*structure pools*) de datos para los mensajes HTTP y las celdas de la tabla hash, así como un módulo de disección HTTP, entre otros. En las siguientes secciones se describe las técnicas propuestas para la disección del tráfico HTTP y su análisis.

#### 2.3.1.1. Emparejando peticiones y respuestas HTTP

Las muestras de tráfico son leídas haciendo uso de la librería **libpcap**, la cual soporta filtrado de paquetes mediante el interfaz Berkeley Packet Filter (BPF) [22]. Por defecto, un filtro deja pasar únicamente las peticiones y respuestas HTTP a la siguiente etapa del procesado. Específicamente, el filtro comprueba si la carga útil (*payload*) del paquete TCP, es decir, el inicio del potencial paquete HTTP, empieza por cualquiera de las cadenas que indican, o bien un verbo de las peticiones HTTP (por ejemplo, GET, POST, HEAD, etcétera), o la cadena literal “HTTP”, en el caso de las respuestas. Este filtro no asegura que el paquete pertenezca a una transacción HTTP, y puede ser reemplazado por otro de acuerdo a las necesidades del usuario de la herramienta *HTTPanalyzer*. Por tanto, la herramienta también realiza una comprobación del formato del paquete durante el proceso de disección.

*HTTPanalyzer* produce registros en un formato que permite el posterior cálculo de estadísticas de calidad de servicio tales como: listas de recursos más solicitados, histogramas de los códigos de respuesta, o la distribución de los tiempos de respuesta, entre otros. Un ejemplo de los registros que *HTTPanalyzer* provee se muestra a continuación.

```
123.111.50.23 | 2311 | 214.223.22.6 | 80 |
1393978285.777375000 | 1393978285.881505000 | 0.104130000 |
OK | 200 | GET | Mozilla/4.0 | service.host.com | /images/icon.gif
```

Con el siguiente formato:

```
IP del cliente; puerto del cliente; IP del servidor; puerto del servidor;
marca temporal de la petición; marca temporal de la respuesta; tiempo de respuesta;
mensaje de respuesta; código de respuesta; método; agente; anfitrión; URI[1]
```

[1]Identificador de recursos uniforme (*Uniform Resource Identifier*)

Un punto clave del método de disección escogido es que *HTTPanalyzer* no requiere el reensamblado ni la reconstrucción de la conexión TCP, y además, sólo el primer paquete de la petición HTTP es considerado para el emparejamiento con su correspondiente respuesta HTTP. Por tanto, sólo el primer paquete de la respuesta es considerado para la obtención del registro de la transacción. Por consiguiente, el tiempo de respuesta calculado está definido como el lapso de tiempo transcurrido entre el primer paquete de la petición HTTP y el primer paquete de la respuesta HTTP.

De esta forma, se consigue una velocidad de procesamiento notable de unos 1.5 Mpps haciendo uso de una única instancia de *HTTPanalyzer*. Tras el filtrado de los paquetes, las peticiones y respuestas HTTP son extraídas y guardadas en una tabla de correlación. Gracias a una función hash, las peticiones y respuestas son guardadas en sus correspondientes celdas de la tabla hash, de acuerdo a la transacción a la que pertenecen.

Después, una vez en la tabla, los mensajes HTTP aguardan la llegada de su correspondiente petición o respuesta, y, una vez esto ocurre, el registro de la transacción HTTP es producido en tiempo real.

#### 2.3.1.2. Mecanismo de *hashing*

En lo que sigue, se procede a desarrollar en mayor detalle el mecanismo de emparejamiento diseñado, el cual es la piedra angular tanto para el procesamiento a alta velocidad como para el balanceo del tráfico HTTP durante el procesamiento en paralelo de este. El diseño de la función hash es complejo y afecta al número de colisiones durante el procesamiento de mensajes HTTP así como al balanceo uniforme de la carga procesada.

Cuando se almacenan los mensajes HTTP en la herramienta *HTTPanalyzer*, resulta necesario evitar las colisiones y hacer más eficiente el uso de la tabla hash. Para ello, es fundamental una función hash uniforme que tenga en cuenta también las restricciones de velocidad impuestas. Como ya se ha señalado, esta función hash no sólo es utilizada para la tabla hash sino también para dividir de forma uniforme los paquetes entre los múltiples consumidores en el escenario de procesamiento en paralelo. Por tanto, uniformidad y aleatoriedad son factores clave para la selección de una función hash adecuada.

$$ValorHash = IPOrigen \oplus PuertoOrigen \oplus IPDestino \oplus PuertoDestino \tag{2.1}$$

Ecuación 2.1: Definición de la función hash basada en la 4-Tupla TCP.

En cuanto a la **uniformidad**, el objetivo es obtener la misma probabilidad para cada valor dado por la función hash, reduciendo por tanto el número potencial de colisiones. Por otro lado, la **aleatoriedad** [23] sirve para la distribución de la carga entre los consumidores, antes de la aplicación de la función módulo para determinar dónde enviar el paquete a procesar. En realidad, si se pretende repartir los paquetes entre dos consumidores, sólo el bit menos significativo del valor hash será relevante. Es decir, la probabilidad de que ese bit sea 0 o 1 deberá ser cercana al 50%. De no ser así, el flujo de paquetes resultaría desequilibrado.

En términos generales, sólo la 4-Tupla (ver la ecuación 2.1) es utilizada como clave hash con el fin de balancear los flujos TCP, asegurando de esta forma que los paquetes pertenecientes al mismo flujo TCP serán distribuidos al mismo consumidor. El problema de esta aproximación es la muy probable variación entre el número de paquetes de un flujo y otro, ocasionando un reparto desigual de los paquetes, y con ello un incremento del número de colisiones en el almacenamiento de los valores en las tablas hash.

A pesar de que la mencionada función hash provee uniformidad, ya que asigna un valor de salida a cada combinación de valores de entrada con la misma probabilidad, los valores reales de entrada, tales como direcciones IP y puertos, no están uniformemente distribuidos en los conjuntos de datos reales. Por ejemplo, José Luis García Dorado et alii, concluyen [24] que estos conjuntos de datos siguen una distribución Zipf. Es más, como W. Shi et al. demostraron [25], debido a que los flujos IP siguen una distribución tipo Zipf, *"a hash-based scheme is not able to achieve load balancing in parallel when the data follows a Zipf-like distribution"*.

#### 2.3.1.3. Reduciendo las colisiones en la tabla hash

Para conseguir emparejar las peticiones y respuestas HTTP no es necesario garantizar que todos los paquetes de un mismo flujo TCP sean enviados al mismo consumidor. Es suficiente con garantizar que al menos la petición y su correspondiente respuesta HTTP sean enviados al mismo consumidor. No es necesario tampoco guardar los mensajes HTTP basándonos en el flujo TCP del que provienen, sino que podemos hacerlo en base a la transacción HTTP a la que pertenecen.

Por tanto, la técnica propuesta elude esta cuestión mediante una función similar a la ecuación 2.1, pero haciendo uso del número de asentimiento (ACK) o del número de secuencia (SEQ) de los paquetes según proceda. Esta nueva función hash (ver Ecuación 2.2) garantiza que los mensajes HTTP pertenecientes a la misma transacción serán almacenados en la misma celda de la tabla hash y distribuidos uniformemente.

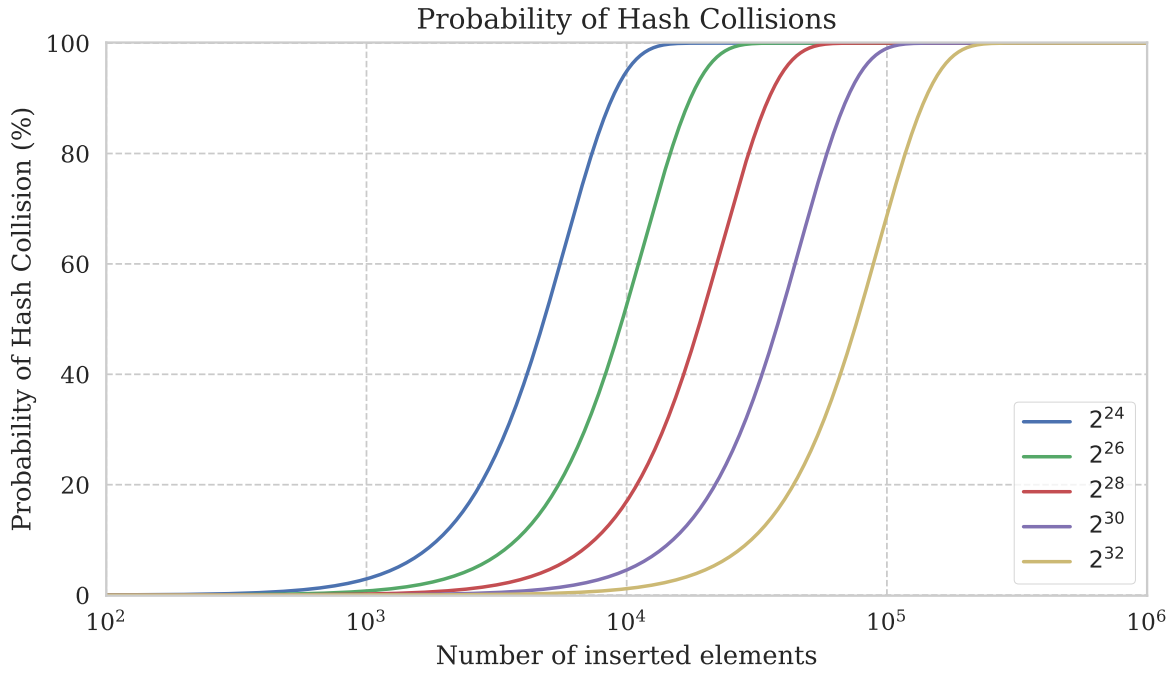



Figura 2.3: Probabilidad de colisiones según el número de elementos insertados para distintos tamaños de tabla.

El tamaño de la tabla hash también será un factor determinante en la probabilidad de colisiones de la tabla. Basándonos en el problema del cumpleaños, se puede estudiar como la probabilidad de colisión se incrementa a medida que el número de elementos introducidos en la tabla crece y también se ve afectado por el tamaño de la tabla.

Por ejemplo, a partir de la ecuación 2.3, dada una habitación con 30 (k) personas, la probabilidad de que 2 de ellas compartan la fecha cumpleaños (con N igual a los 365 días que hay en un año) es del 70 %. Al incrementar este número a 50 personas, la probabilidad asciende al 97 %.

Como puede apreciarse en la figura 2.3, el número de colisiones disminuye cuanto mayor es la tabla. Emplear una tabla mayor no incrementará necesariamente el uso real de los recursos, ya que, independientemente del tamaño de la tabla hash, el número de elementos introducidos siempre será mismo para una misma entrada de datos, y por

$$Valor\ Hash = \begin{cases} Petición: & IP\ Or. \oplus Puerto\ Or. \oplus IP\ Dest. \oplus Puerto\ Dest. \oplus ACK \\ Respuesta: & IP\ Or. \oplus Puerto\ Or. \oplus IP\ Dest. \oplus Puerto\ Dest. \oplus SEC \end{cases} \tag{2.2}$$

Ecuación 2.2: Definición de la nueva función hash.

$$1 - e^{\frac{-k(k-1)}{2N}} \tag{2.3}$$

Ecuación 2.3: Probabilidad de la ocurrencia de colisiones

tanto, a pesar de utilizar una tabla hash enorme, la mayor parte de esta permanecerá en memoria virtual y los accesos, independientemente del tamaño de la tabla hash, seguirán siendo aleatorios. Además, en este caso las celdas de la tabla hash se irán liberando y reocupando según lleguen los mensajes HTTP y sus mensajes complementarios. Cada vez que se imprime una transacción, una celda (si no hay colisión) es liberada y por tanto el uso de la tabla no es siempre creciente durante la ejecución de *HTTPanalyzer*.

#### 2.3.1.4. Balanceo de carga a nivel de conexión Sub-TCP

Para la ejecución paralela de varios consumidores *HTTPanalyzer* se utilizó una herramienta balanceadora llamada *PacketFeeder* la cual distribuye los paquetes entre las instancias de *HTTPanalyzer*, leyendo los paquetes del búfer de datos del controlador de la tarjeta de red y compartiendo una región de memoria con los consumidores.

Para cada paquete entrante, un número hash es calculado utilizando las cabeceras del paquete y, entonces, la operación módulo es aplicada con el fin de elegir el consumidor al que será enviado el paquete en cuestión. Utilizar la función genérica basada en la 4-Tupla (Ecuación 2.1) aseguraría que los paquetes de la misma conexión acabasen en el mismo consumidor *HTTPanalyzer*. Sin embargo, como se apuntó previamente, este método podría conducir a un comportamiento desequilibrado siempre que algunas conexiones tuvieran muchos más paquetes y transacciones que otras.

En consecuencia, se hace uso de una función similar a la descrita en la ecuación 2.2, pero con el fin de conseguir una mejor aleatorización del bit menos significativo del valor hash, al resultado de las anteriores operaciones también le aplicamos (al valor hash) la disyunción exclusiva (XOR) byte a byte del número de secuencia o de asentimiento (según

$$Consumidor = \begin{cases} Petición: & IP\,Or. \oplus Puerto\,Or. \oplus IP\,Dest. \oplus Puerto\,Dest. \\ & \oplus ACK \oplus (Ack_1 \oplus Ack_2 \oplus Ack_3 \oplus Ack_4) \\ Respuesta: & IP\,Or. \oplus Puerto\,Or. \oplus IP\,Dest. \oplus Puerto\,Dest. \\ & \oplus SEQ \oplus (Seq_1 \oplus Seq_2 \oplus Seq_3 \oplus Seq_4) \end{cases} mod.\, n \tag{2.4}$$

Ecuación 2.4: Función hash para el reparto de carga. Los subíndices indican el correspondiente byte del número de asentimiento (ACK) o del número de secuencia (SEQ).

proceda). Después, se toma el resto de dividir este valor entre el número de consumidores (n), el cual resulta en el número del consumidor destinatario.

Como resultado, se obtiene la función hash descrita en la ecuación 2.4, la cual garantiza que tanto los consumidores recibirán la misma carga de paquetes y que los mensajes HTTP de la misma transacción terminarán en el mismo consumidor. En la sección 2.4 se discute en detalle los resultados de la función hash propuesta y hasta qué punto distribuye los valores hash producidos.

#### 2.3.1.5. Procesamiento de los paquetes

Tal como se observa en la figura 2.4, una vez la petición o respuesta HTTP llega, el valor hash es calculado haciendo uso de la 4-Tupla formada por la IP y el puerto de origen; la IP y puerto de destino; así como el número de asentimiento o de secuencia dependiendo de si se trata de una petición o una respuesta, respectivamente (Ecuación 2.2).

Este valor es usado para encontrar la celda adecuada en la tabla hash, dividiendo el valor entre el tamaño de la tabla y tomando el resto. La principal condición para emparejar una petición HTTP con su correspondiente respuesta HTTP es que ambas deben coincidir en su 4-Tupla (IP origen, puerto origen, IP destino, puerto destino) y además, **la respuesta HTTP debe tener un número de secuencia igual al número de asentimiento de la petición HTTP**.

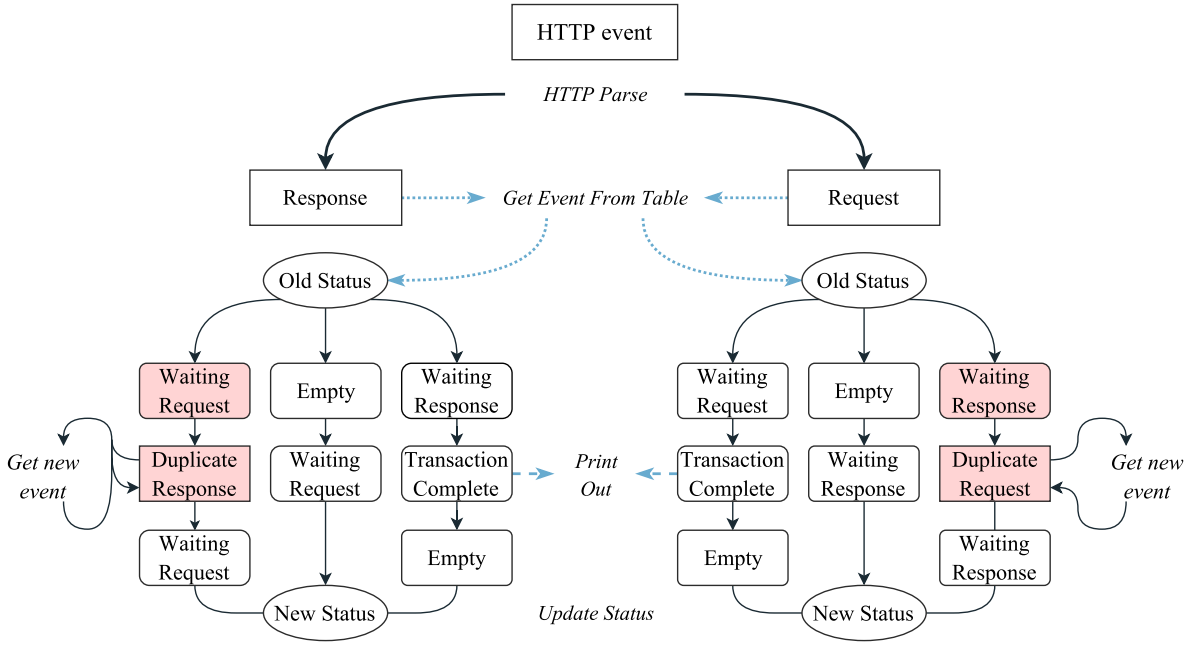



Figura 2.4: Diagrama del procesamiento de un mensaje HTTP.

Posteriormente, surgen distintas posibilidades dependiendo de si la celda destino está vacía sin ninguna contraparte esperando, o si por el contrario, su mensaje pretendiente ha tomado ya la celda destino y por tanto está esperando el mensaje complementario. Sin embargo, hay un tercer escenario, (mostrado en rojo en la figura 2.4) en el cual un mensaje duplicado ya ha sido almacenado en la tabla, siendo este mensaje bien una petición o una respuesta que ha llegado previamente. Principalmente, estos casos pueden ser resultado de posibles retransmisiones o duplicados, pero existe un caso especial que ocurre con las respuestas con mensaje *100 Continue*, las cuales usualmente suceden en el transcurso de una petición de tipo POST, en la cual se están enviando datos al servidor.

Estas largas peticiones normalmente terminan con un código de respuesta (200 OK en caso satisfactorio) al final de la transacción. Estos duplicados se almacenan en la tabla como colisiones en busca su correspondiente contraparte también retransmitida o duplicada. En caso de que jamás llegue una contraparte, estas serán limpiadas de la tabla hash mediante un recolector de basura.

### 2.3.2. Limitaciones resultado del conocimiento parcial de la conexión TCP

Es necesario destacar que el procedimiento anteriormente descrito no es tan preciso como la reconstrucción y reensamblado de los flujos TCP debido al posible desorden de paquetes y retransmisiones.

En concreto, se está haciendo uso de conocimiento parcial de las conexiones TCP y de las inmediaciones de cada transacción HTTP, y no del conocimiento global de la conexión TCP completa. Mientras que esto supone una ventaja en velocidad, hay por supuesto limitaciones para extraer con precisión las peticiones y respuestas HTTP de su correspondiente conexión TCP.

Sin embargo, se han empleado varios heurísticos para mitigar las imprecisiones lo máximo posible, los cuales se presentan a continuación.

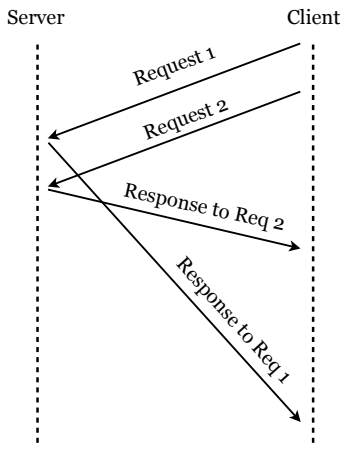



Figura 2.5

Las peticiones y respuestas HTTP pueden llegar fuera de orden.

#### 2.3.2.1. Mensajes HTTP desordenados

En primer lugar, los mensajes HTTP pueden llegar desordenados, implicando que una respuesta correspondiente a una petición pretérita puede llegar después que la respuesta de una petición realizada más recientemente (dentro de la misma conexión TCP) tal como se aprecia en la figura 2.5. En consecuencia, las transacciones HTTP pueden ser impresas por *HTTPanalyzer* fuera de orden. Esto es debido a que las conexiones TCP no son reconstruidas, y por tanto, los segmentos TCP pueden llegar en un orden arbitrario dependiendo de las dinámicas de los paquetes IP durante el encaminamiento (o *enrutamiento*) desde el cliente al servidor. Para soslayar esta limitación el almacenamiento de los mensajes HTTP se realiza sin importar cual de las dos contrapartes (petición o respuesta) ha llegado primero. Por tanto, el emparejamiento puede ocurrir en los dos sentidos.

#### 2.3.2.2. Retransmisiones

Las retransmisiones son más frecuentes que los paquetes fuera de orden. En el evento de un mensaje retransmitido, estos son también almacenados en sus correspondientes celdas, en la lista de colisiones, produciendo registros duplicados. Estos duplicados deberán ser filtrados por el analista atendiendo a aquellas retransmisiones que tengan la misma 4-Tupla y el mismo número de asentimiento o de secuencia.

#### 2.3.2.3. Precisión

Resulta evidente que una precisión y exactitud absoluta en la detección de peticiones y respuestas HTTP no es posible mediante el método propuesto. Sin embargo, el objetivo de este desarrollo es la extracción de estadísticas agregadas que sean útiles en un centro de control de redes (*Network Operations Center*, NOC), sacrificando por tanto precisión por velocidad.

Por ejemplo, como se ha explicado anteriormente, sólo el primer paquete de la petición y de la respuesta son tenidos en cuenta en la evaluación del tiempo de respuesta y los códigos de respuesta. De este modo, el Localizador Uniforme de Recursos (*Uniform Resource Locator*, URL) puede verse truncado si el paquete es mayor que la unidad máxima de transmisión (MTU en sus siglas en inglés), es decir 1,518 bytes. El RFC 2616 (Hypertext Transfer Protocol HTTP/1.1), en su sección 3.2.1 [26] indica que el protocolo HTTP no establece un límite a priori en cuanto a la longitud de una URI.

Los servidores deben ser capaces de manejar la URI de cualquier recurso que ellos sirvan (*"The HTTP protocol does not place any a priori limit on the length of a URI.*

*Servers MUST be able to handle the URI of any resource they serve."*). Aunque la realidad es que la mayoría de los navegadores [27] soportan hasta 80,000 caracteres de media y que el servidor web de Apache tiene un límite de 8,192 caracteres.

Algunos navegadores como Internet Explorer tienen un límite de 2,048 caracteres. Las recomendaciones sobre estilo de URLs indican que las URLs largas no son buenas si los servicios web pretenden ser indexados por los motores de búsqueda, ya que el protocolo *sitemaps* [28] posee un límite de 2,048 caracteres para las URL. Los servicios de posicionamiento en buscadores (*search engine optimization*, SEO) dan menos peso a este tipo de URLs.

En la sección de resultados se mostrará cómo las estadísticas agregadas obtenidas mediante la técnica propuesta son altamente similares a aquellas obtenidas mediante la reconstrucción y el reensamblado de las conexiones TCP.

#### 2.3.2.4. Recolector de Basura

Cabe la posibilidad de que algunas peticiones y respuestas jamás sean eliminadas de nuestra tabla de correlación, si por ejemplo, el mensaje complementario no está presente en el tráfico o fichero analizado, lo cual implica un gasto de recursos y un posible incremento de las colisiones en la tabla. Algo similar ocurre para aquellas respuestas con un gran retraso, cuyas peticiones ocupan recursos durante demasiado tiempo. Ambos efectos comprometen el rendimiento de la herramienta ya que cuanto mayor número de colisiones alberga la tabla hash, mayor tiempo requerirá la búsqueda.

Para mitigar estos efectos, un recolector de basura comprueba el estado de la tabla y recorre las celdas activas de esta, eliminando aquellas transacciones que no muestren cambios durante los últimos 60 segundos de captura. Estos mensajes no emparejados se imprimen junto con el resto de transacciones HTTP debido a que también suponen información valiosa para el analista.

## 2.4. Evaluación de la herramienta *HTTPanalyzer*

En esta sección se presentan los resultados obtenidos con la solución propuesta *HTTPanalyzer* los cuales son comparados con Tshark. Nuestro requisito principal es el rendimiento de la herramienta manteniendo un nivel razonable de precisión en cuanto a las estadísticas HTTP. En primer lugar se discutirán los problemas de precisión encontrados así como su magnitud. En concreto se tratarán las limitaciones referentes a la pérdida de información en las URLs de las peticiones debido a la fragmentación

de los paquetes; los tiempos de respuesta; los códigos de respuesta y los verbos HTTP. Finalmente, se discutirán los resultados de rendimiento de la aplicación.

### 2.4.1. Evaluación de la precisión

A continuación se discute la precisión de HTTPanalyzer para distintas métricas.

#### 2.4.1.1. Pérdida potencial de información en las URLs solicitadas

Para ambas muestras de tráfico (ver la tabla 2.1) se evaluó la cantidad de URLs truncadas por el método empleado en *HTTPanalyzer* y la URL máxima que es capaz de extraer a partir de los paquetes. Después, se compararon con los resultados obtenidos mediante Tshark. En la figura 2.6 puede observarse que *HTTPanalyzer* (círculos verdes) coincide casi idénticamente con los resultados de Tshark (mostrados con triángulos naranjas), exceptuando aquellas URLs mayores a 1,455 caracteres.

Esta longitud es la máxima que *HTTPanalyzer* puede gestionar. Las URLs indicadas están destacadas en el área roja de la figura y suponen tan solo un 0.04% sobre la cantidad total de URLs, considerando ambos ficheros.

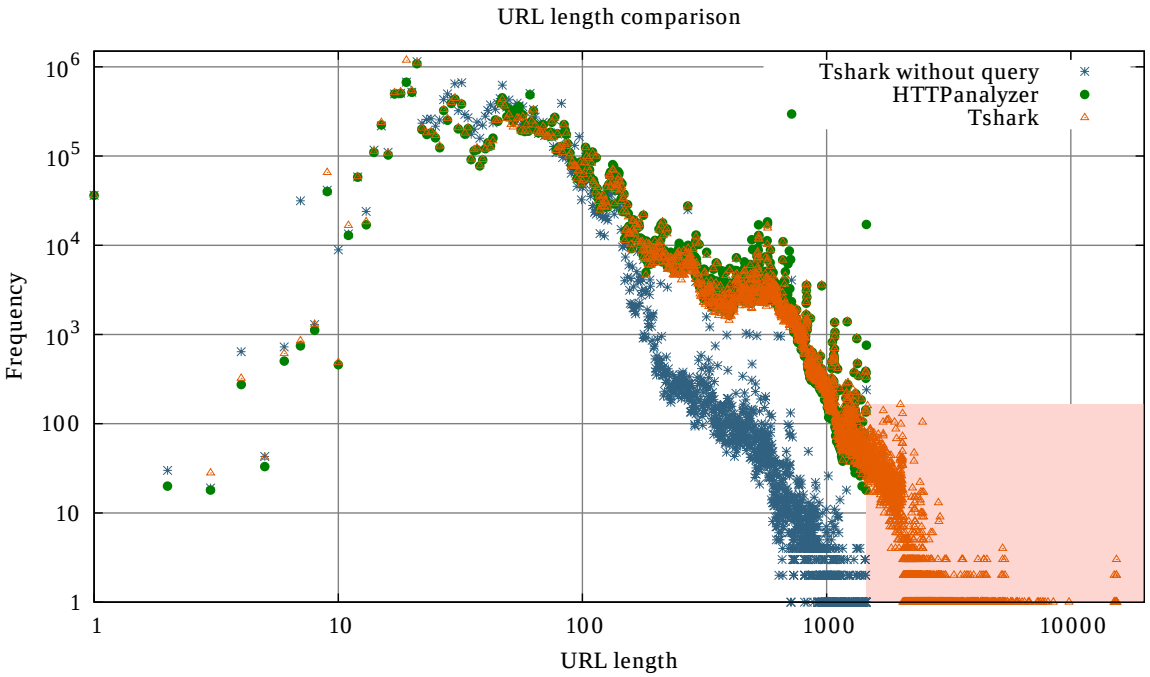



Figura 2.6: Comparación del tamaño de URLs. Las destacadas en el area roja son mayores a la limitación de *HTTPanalyzer* y representan un 0.04% del total de URLs analizadas

Dependiendo del análisis realizado, los parámetros de la consulta HTTP pueden ser relevantes o carecer de importancia. Estos parámetros los constituyen aquellos caracteres de la URL tras el primer carácter *'?'*, los cuales indican parámetros para el recurso solicitado. En la figura 2.6 también se ha dibujado (en asteriscos azules), los resultados de Tshark ignorando los parámetros presentes en la URL de la consulta HTTP. Como puede apreciarse ninguno de ellos excede los 1,455 caracteres que limitan a *HTTPanalyzer*, indicando que la mayoría de los caracteres presentes en una URL los forman los parámetros de la consulta. Se considera que la parte más significativa de una URL está en realidad al principio, indicando el recurso invocado por la solicitud.

En cualquier caso, los registros impresos por *HTTPanalyzer* contienen suficientes parámetros (4-Tupla, tiempos, etc.) como para filtrar a posteriori los paquetes correspondientes a estas URLs largas, y eventualmente, realizar un análisis manual de estas, si así fuese necesario.

#### 2.4.1.2. Tiempo de Respuesta

El tiempo de respuesta es uno de los parámetros HTTP más interesantes para la medición de la calidad de servicio, el cual sirve para detectar posibles degradaciones súbitas de los servicios web. Se ha comparado la función de distribución acumulada inversa (*Complementary cumulative distribution function*, CCDF) de los tiempos de respuesta proporcionados tanto por Tshark como por *HTTPanalyzer*.

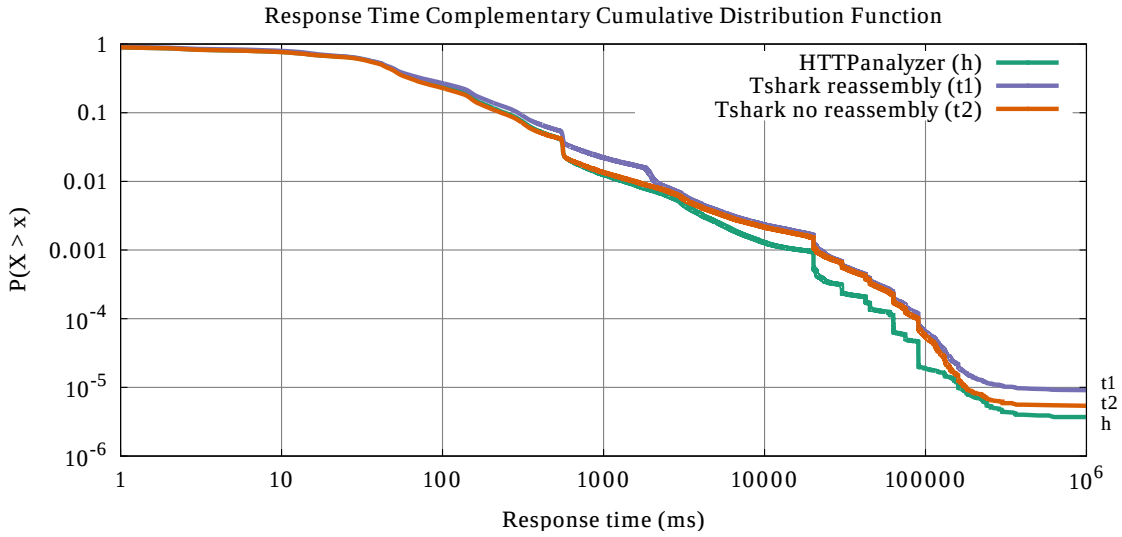



Figura 2.7: Comparación de la precisión de *HTTPanalyzer* vs. Tshark en el cálculo de la CCDF de los tiempos de respuesta de las transacciones HTTP.

*HTTPanalyzer* mide el tiempo de respuesta entendiéndolo como la diferencia entre la marca temporal del primer paquete de la petición HTTP y la marca temporal del primer paquete de la respuesta HTTP. Sin embargo, generalmente Tshark mide el tiempo de respuesta como el intervalo entre la primera petición y el último paquete de la respuesta. No obstante, Tshark puede medir el tiempo de respuesta de una forma diferente cuando el reensamblado TCP está desactivado usando únicamente los primeros paquetes tal como hace *HTTPanalyzer*. Con el fin de realizar una comparación justa, en la figura 2.7 se presentan tanto los resultados de sendos métodos de Tshark como los de *HTTPanalyzer*, no pudiéndose apreciar una diferencia significativa entre ellos.

#### 2.4.1.3. Códigos de respuesta

La evaluación de los códigos de respuesta es de una importancia fundamental para la búsqueda de errores. Por ejemplo, una gran cantidad de errores 404 (No encontrado) en los códigos de respuesta HTTP implica que en el servicio web analizado hay una gran cantidad (o unos pocos que son muy solicitados) de enlaces muertos o recursos desaparecidos. En un caso análogo, los errores 5xx puede resultar de gran interés, por

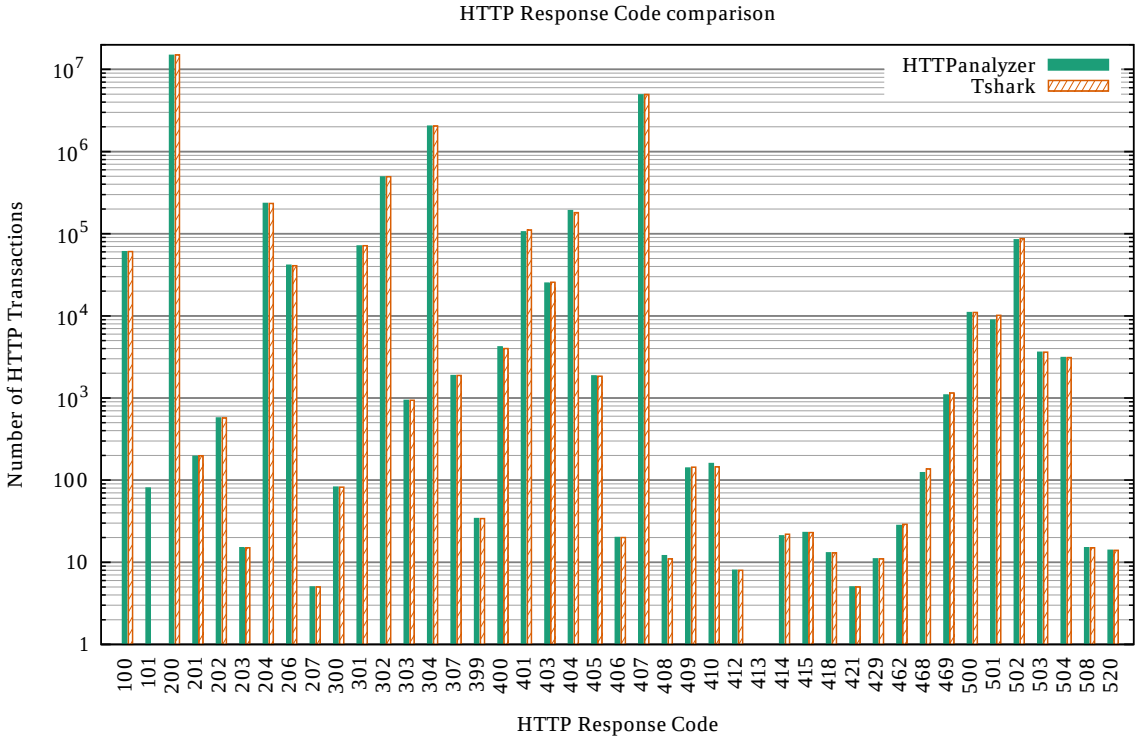



Figura 2.8: Comparación de la precisión otorgada por *HTTPanalyzer* versus Tshark en el conteo de los códigos de respuesta de las transacciones HTTP.

ejemplo, los errores con código 500 (Error Interno) pueden ser reportados por recursos web dinámicos en caso de un error en los objetos dinámicos invocados.

Como se aprecia en la figura 2.8, el conteo de los códigos de respuesta de *HTTPanalyzer* es prácticamente idéntico al de Tshark y la diferencia media con los resultados de Tshark en ciertos códigos de respuesta específicos es de un 2.6% con una mediana de 1.3%. Esta diferencia es debida a algunas pérdidas en *HTTPanalyzer* cuando se envían varias consultas HTTP en ráfaga (*pipelined*) en el mismo flujo.

#### 2.4.1.4. Métodos o Verbos HTTP

Una comparación similar puede hacerse con las otras métricas de las transacciones HTTP, como por ejemplo el histograma de los códigos de métodos HTTP que se muestra en la figura 2.9, el cual muestra que la herramienta *HTTPanalyzer* provee aproximadamente el mismo resultado que el otorgado por Tshark. Algunas diferencias menores como el conteo de los verbos PUT se deben a la pérdida de transacciones en los *bordes* entre trozo y trozo de los ficheros. Como se explicó anteriormente, los ficheros originales de la tabla 2.1 fueron troceados para poder ser procesados con Tshark. De otra forma los ficheros serían demasiado grandes para que Tshark pueda manejarlos.

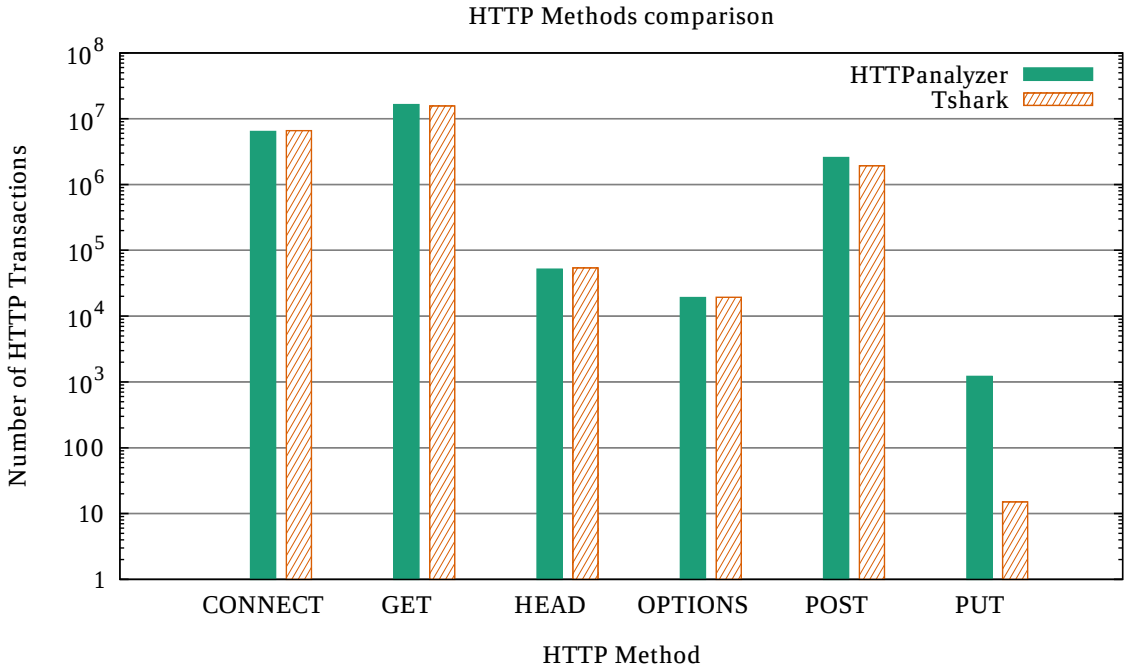



Figura 2.9: Comparación de la precisión otorgada por *HTTPanalyzer* versus Tshark en el conteo de los verbos HTTP.

### 2.4.2. Pruebas de rendimiento

Para esta sección, los experimentos de rendimiento también fueron realizados con los ficheros de la tabla 2.1, en dos escenarios diferentes (ver figuras 2.1 y 2.2. En primer lugar se llevó a cabo una prueba fuera de línea (*offline*) con el fin de comprobar la capacidad de procesar ficheros en sistemas de almacenamiento de alta velocidad (10 Gbps) mediante una única instancia de *HTTPanalyzer*. Estos sistemas de almacenamiento son comunes en sondas de análisis del tráfico donde se requiere gran velocidad de almacenamiento (y capacidad) para la captura (escritura) y análisis (lectura) de datos.

Seguidamente, se comprobó el rendimiento de *HTTPanalyzer* en el procesamiento de tráfico a 20 Gbps mediante tráfico enviado en línea (*online*) mediante un emisor de tráfico alojado en una máquina y su correspondiente recepción en otra máquina, la cual incorpora la herramienta balanceadora de tráfico *PacketFeeder* con el propósito de distribuir el tráfico entrante entre las instancias de *HTTPanalyzer*, haciendo uso de la función hash uniforme explicada en la sección 2.3.

No obstante, con el fin de explicar mejor estos resultados, a continuación se profundiza en los detalles de la función hash utilizada tanto para distribuir los paquetes en la tabla hash de *HTTPanalyzer* como para hacerlo entre los consumidores.

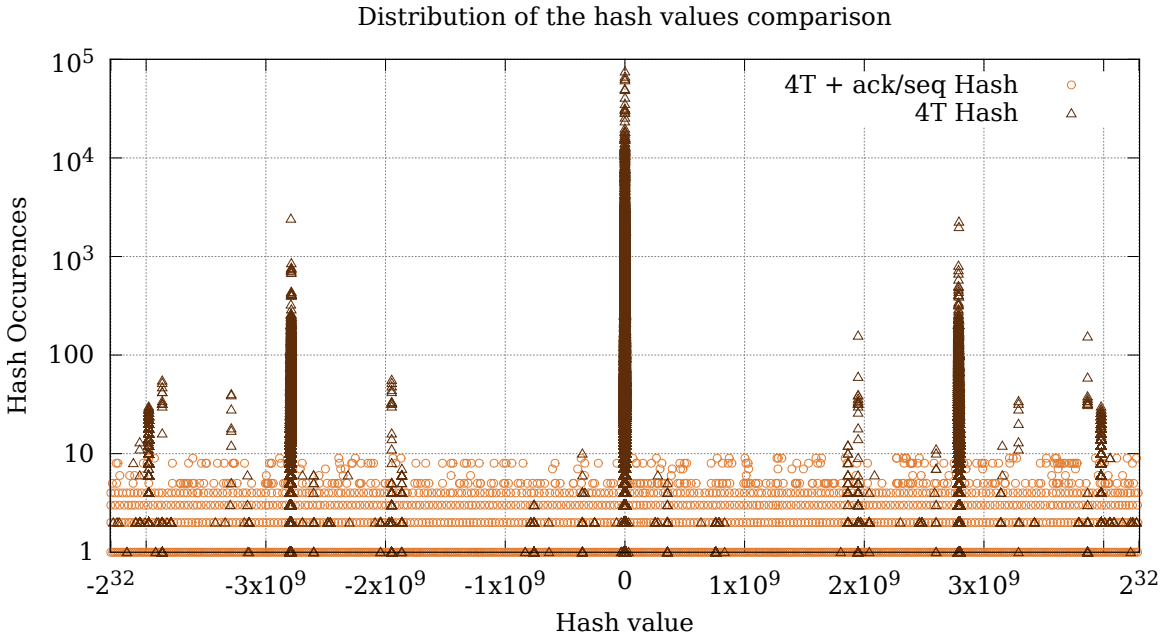



Figura 2.10: Comparación de la distribución de valores de diferentes funciones hash.

#### 2.4.2.1. Pruebas de la función hash

El histograma de los valores hash es la figura clave para explicar la selección de la función hash realizada, ya que resume, en un único gráfico, si los valores de la función hash son uniformes para un balanceo de los paquetes equitativo. Dicho lo cual, la figura 2.10 muestra el histograma de valores para las ecuaciones 2.1 y 2.2 explicadas en la sección 2.3.1.2. Esta distribución está dividida en dos lados, con números negativos a un lado y positivos al otro, indicando que el lado negativo del eje de las abscisas corresponde con uno de los dos consumidores y el lado positivo con el otro consumidor. Cada punto representa el número de ocurrencias para un valor hash específico.

Resulta interesante y notorio indicar que los números de 32 bit que forman el número de secuencia (o de asentimiento) TCP, aleatorizan el valor hash resultante, reduciendo el número de colisiones (ocurrencias de un mismo valor hash) sin afectar al emparejamiento de los mensajes HTTP (Ver la función hash explicada en la sección 2.3.1.2, ecuación 2.2).

Como muestra la figura 2.10, las colisiones se ven reducidas en varios órdenes de magnitud cuando se hace uso de los números de secuencia y asentimiento, ya que estos números de 32 bit aleatorizan todo el valor hash. Estos son inicializados aleatoriamente por la pila TCP cuando las conexiones son creadas. Esta distribución (mostrada en marrón clarito con círculos) es la misma para la ecuación 2.2 y 2.4. No obstante, la ecuación 2.4 es usada para los experimentos con varios consumidores, mientras que la ecuación 2.2 se emplea en la tabla hash de *HTTPanalyzer*. Es necesario indicar que en la tabla hash no hay necesidad de aleatorizar al máximo el bit menos significativo y en consecuencia, se reducen los requisitos de procesamiento para computar este valor.

Tras lo expuesto en la discusión anterior, se procede a la presentación de los resultados de rendimiento para los escenarios fuera de línea (*offline*) y en línea (*online*) representados en las figuras 2.1 y 2.2.

### 2.4.3. Pruebas con un único consumidor

El siguiente experimento pretende demostrar la capacidad de *HTTPanalyzer* para diseccionar ficheros PCAP a 10 Gbps. La figura 2.1 detalla este escenario. Para esta prueba se utilizó una máquina con un Intel Xeon E3-1230 v3 @ 3.30Ghz con 32GB de RAM y un sistema de almacenamiento de alta velocidad formado por un RAID 0 de 8 discos SSD Samsung 840 con velocidades de lectura por encima de los 10 Gbps.

Las pruebas se realizaron mediante los ficheros de tráfico descritos en la tabla 2.1. También se realizó una prueba de velocidad con un trozo de fichero de 15GB almacenado

Cuadro 2.2: Pruebas de velocidad con una instancia de HTTPanalyzer.

| Sistema de almacenamiento | Velocidad (Gbps) | Velocidad (Mpps) |
|---|---|---|
| RAID 0 | 10.6 ±0.58 | 1.8 ±0.15 |
| RAM | 13.8 ±1.4 | 2.1 ±0.26 |

en un sistema de ficheros en RAM con el propósito de medir la máxima velocidad de procesamiento de nuestra herramienta. Estas pruebas dieron unos resultados satisfactorios, mostrados en la tabla 2.2, probando que una única instancia de *HTTPanalyzer* es capaz de procesar más de 10 Gbps de tráfico.

### 2.4.4. Pruebas con múltiples consumidores

En esta sección se discuten los resultados de las pruebas efectuadas mediante múltiples consumidores de *HTTPanalyzer* para el procesamiento de 20 Gbps (dos flujos de 10 Gbps) de tráfico en línea en una única máquina. El objetivo es demostrar que múltiples instancias de *HTTPanalyzer* pueden operar en paralelo con una carga similar gracias a la función hash propuesta. Para realizar el experimento, dos máquinas fueron dispuestas en la forma descrita en la figura 2.2. La máquina A es la misma utilizada en el anterior escenario, pero esta vez, el tráfico almacenado en el sistema de almacenamiento de alta velocidad es enviado mediante una reproductora de tráfico NetFPGA [29] a través de dos enlaces ópticos 10GbE, enviando los mismos datos por cada uno de los cables. Este generador de tráfico a 10Gbps [30] forma parte de un entorno de pruebas del proyecto europeo Fed4Fire[2], y es capaz de enviar tráfico a 10Gbps por enlace.

Inmediatamente después, la máquina B recibe el tráfico haciendo uso de HPCAP[3] [31], un driver a nivel de núcleo diseñado para tarjetas de red (*NIC*) Intel capaz de procesar un enlace 10GbE totalmente saturado. Ya que este driver lee los paquetes desde cada interfaz de forma independiente, dos instancias del *PacketFeeder* fueron utilizadas, una para cada línea de 10GbE.

Así mismo, dos consumidores *HTTPanalyzer* fueron dispuestos en cada una de las instancias de *PacketFeeder*. Esto hace un total de cuatro *HTTPanalyzer* ejecutándose en paralelo, cada uno asignado a un núcleo de procesamiento distinto. Cada *PacketFeeder* distribuye los paquetes que recibe de su correspondiente enlace de red haciendo uso del método de reparto explicado en las secciones anteriores, lo cual asegura una distribución uniforme de los paquetes y de las transacciones HTTP en cada consumidor.

[2] `https://fed4fire.eu`

[3] `https://github.com/hpcn-uam/HPCAP`

Cuadro 2.3: Resultados de los experimentos con múltiples instancias de *HTTPanalyzer*

| | Consumidor A-1 | Consumidor A-2 | Consumidor B-1 | Consumidor B-2 |
|---|---|---|---|---|
| Paquetes Recibidos | 49.86% | 50.02% | 49.94% | 50.02% |
| Transacciones HTTP | 50.01% | 49.98% | 50.01% | 49.98% |

De manera interesante, las cuatro instancias recibieron prácticamente la misma cantidad de tráfico, tal y como se aprecia en la tabla 2.3. Los resultados indican que el método propuesto para el balanceo de carga es muy efectivo. Los experimentos a velocidades de 40GbE no pudieron ser realizados ya que esta tecnología es, en el momento de esta investigación, minoritaria y cara, así como a las limitaciones del reproductor de tráfico. Sin embargo, estos resultados se muestran prometedores y hacen pensar que *HTTPanalyzer* pueda hacer frente a tasas mayores usando la misma aproximación consistente en distribuir la carga entre múltiples instancias de *HTTPanalyzer*.

### 2.4.5. Comparación de rendimiento contra Tshark

Para completar la evaluación del rendimiento de *HTTPanalyzer*, se realizó una comparación de las velocidades de procesamiento de *HTTPanalyzer* versus Tshark. A pesar de que Tshark provee mayor nivel de detalle en las métricas y estadísticas del tráfico HTTP, resulta evidente que no es capaz de hacer frente a un análisis en tiempo real a altas tasas. La figura 2.11 muestra la velocidad de procesamiento de *HTTPanalyzer* y Tshark. Puede observarse que *HTTPanalyzer* es 43 veces más rápido que Tshark. El experimento de medición fue realizado fuera de línea (*offline*), leyendo los archivos desde un sistema de ficheros en RAM, tal como se hizo en la sección 2.4.3.

Adicionalmente, la figura 2.12 ilustra las pérdidas de paquetes sufridas por Tshark en la recepción de tráfico a 10 Gbps durante un experimento en línea similar al realizado en la sección 2.4.4. Claramente, la pérdida de paquetes resulta muy significativa, lo cual reafirma que Tshark no es adecuado para el procesamiento en línea en escenarios con tasas de varios Gbps de tráfico. Es claro que existe un compromiso entre la precisión otorgada por Tshark y la velocidad de *HTTPanalyzer*. No obstante, la precisión que ofrece *HTTPanalyzer* resulta notable, (como se aprecia en la sección 2.4.1), la cual, junto con las limitaciones de Tshark presentadas en esta sección, hacen de *HTTPanalyzer* la elección perfecta para el análisis en tiempo real y a altas velocidades del tráfico HTTP.

## 2.5. Conclusiones sobre *HTTPanalyzer*

En esta investigación, publicada en *Computer Networks* bajo el título *Multi-Gbps HTTP Traffic Analysis in Commodity Hardware Based on Local Knowledge of TCP Streams* [32], se ha propuesto *HTTPanalyzer*, un sistema de alto rendimiento para el análisis de tráfico HTTP, el cual consigue una velocidad de 10 Gbps. Se obtiene también un excepcional rendimiento de 20 Gbps en experimentos realizados en línea con múltiples consumidores gracias a la función hash propuesta en este trabajo. Además, los resultados se obtuvieron en *hardware* convencional, sin necesidad de procesadores *ad-hoc* de alta velocidad ni mediante dispositivos paralelos masivos, diferenciándose del estado del arte.

Finalmente, *HTTPanalyzer* provee información en tiempo real para la medición de la calidad de servicio del tráfico web en grandes organizaciones. Estas métricas tienen una importancia estratégica debido a su cercana relación con la calidad de experiencia del usuario final, permitiendo detectar cambios en los comportamientos de los servicios web.

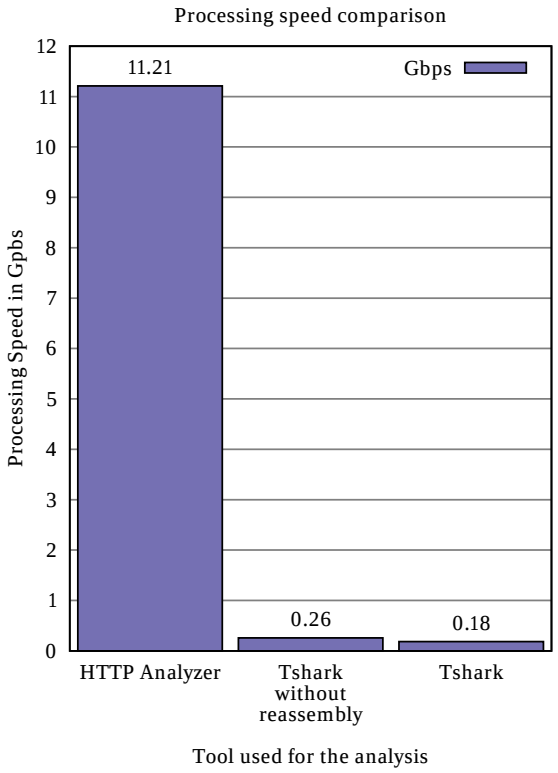



Figura 2.11
Velocidad de procesamiento de Tshark y *HTTPanalyzer*

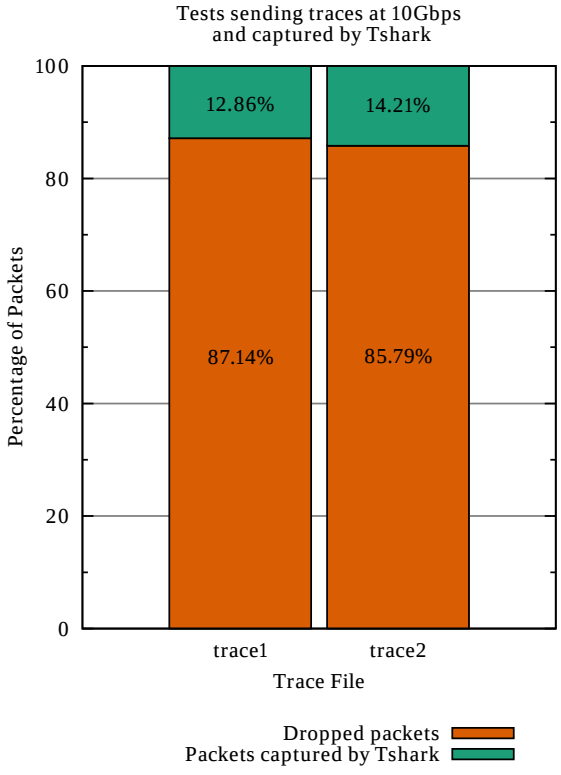



Figura 2.12
Porcentaje de pérdida de paquetes por parte de Tshark durante la recepción de tráfico en línea.

Pruebas de rendimiento. Tshark versus *HTTPanalyzer*


La Princesa Sabia (*The Wise Princess*) es un cuento de hadas británico escrito por Mary De Morgan sobre una princesa que lo sabe todo, excepto la verdadera felicidad. Ilustración de Walter Crane. 1880

CAPÍTULO 3

# *Loginson*: Recolección y Análisis de Registros

Hoy en día la mayoría de los sistemas y aplicaciones generan registros de eventos (de ahora en adelante *logs*) que resultan esenciales desde el punto de vista de la seguridad y la monitorización de los sistemas, así como para la depuración y diagnóstico de problemas de programación, comprobar el estado de los sistemas, detectar problemas de configuración o incluso ataques a sistemas. Con este propósito, resulta necesario un repositorio de *logs* en el cual los *logs* puedan ser consultados y visualizados rápidamente.

Esta investigación presenta *Loginson*, un sistema de centralización de alto rendimiento para la recolección y procesamiento de *logs* a gran escala en grandes infraestructuras de TI (Tecnologías de la Información). A parte de la recolección de *logs*, *Loginson* provee analíticas de alto nivel a través de una interfaz visual con el fin de favorecer la resolución de incidentes críticos.

Es necesario remarcar que, en el momento de la realización de esta investigación, *Loginson* sobrepasa el rendimiento de las soluciones similares presentes en el estado del arte, aprovechando el escalado vertical del sistema, y por tanto reduciendo las inversiones de capital (*Capital Expenditure*, CAPEX) y los costes operacionales (*Operational Expense*, OPEX) de los despliegues con un gran volumen de registros.

## 3.1. Introducción a los sistemas ETL

En grandes centros de datos y, en general, grandes infraestructuras de TI, la monitorización resulta una actividad esencial para asegurar la estabilidad y la calidad de servicio [33]. Con este fin, los sistemas y las aplicaciones generan *logs*, los cuales proveen valiosa información acerca del sistema y el estado de las aplicaciones. Por ejemplo, los *logs* de sistema sirven para detectar un alto uso de memoria real o de intercambio (*swap*) en un nodo específico del centro de datos, lo cual es de gran utilidad para la señalización de una posible anomalía en el comportamiento del nodo.

Esta investigación se enfoca en la recolección y procesamiento de *logs* en grandes infraestructuras. Las características distintivas de este caso de uso, comparadas con los sistemas genéricos de recolección (Hadoop[1] [34], HBase[2] [36], Elasticsearch[3] [37], etc. [38]), son dos. Primero, los *logs* son producidos por un gran número de servidores y routers, resultando en una cantidad de datos enorme. Estos son producidos a altas tasas, lo cual supone un serio reto para la recolección de *logs*. En segundo lugar, la inspección de los *logs* requiere dos fases secuenciales: el vistazo de alto nivel y, entonces, el análisis detallado de determinadas partes de los *logs*. En realidad, en primera instancia, un resumen de alto nivel debe ser proporcionado, por ejemplo, con un gráfico mostrando la evolución de diferentes códigos de respuesta en el tiempo. Cuando quiera que una anomalía ocurra, debido por ejemplo a un incremento en el número de errores internos (500), el administrador deberá ser capaz de inspeccionar en detalle los *logs* en busca de la causa raíz del problema. No obstante, **esta segunda fase sólo se aplica a aquellos datos dentro del intervalo de tiempo donde la anomalía ha ocurrido**. La naturaleza de este sistema bi-fase para la inspección de los *logs* requiere un sistema de procesamiento distribuido y un sistema de almacenamiento específicamente diseñado para este caso de uso, el cual es el epicentro de esta investigación.

En realidad, en esta investigación se propone un nuevo sistema de procesamiento y almacenamiento llamado *Loginson*, el cual está específicamente orientado al procesamiento, almacenamiento y visualización de cantidades masivas de datos. Esto sigue la metodología de extracción, transformación y carga (ETL, *Extract Transform and Load* en sus siglas en inglés) de datos. Como es lógico, la detección de incidentes entre millones de mensajes de *log* resulta todo un reto, el cual requiere un sistema de alto rendimiento que provea las capacidades anteriormente mencionadas para una inspección en dos etapas:

[1] `http://hadoop.apache.org/`
[2] `https://hbase.apache.org/`
[3] `https://www.elastic.co/`

alto nivel y detallada. En este punto, antes de entrar en los detalles de la arquitectura de *Loginson*, se explicará la motivación detrás de esta investigación: proveer al administrador de redes y sistemas con un sistema potente, escalable y fácil de utilizar, y el cual sea capaz de hacer frente al reto del análisis de *logs*.

### 3.1.1. Motivación

El actual Internet muy probablemente crecerá [39][40][41] y evolucionará hacia el llamado Internet de las cosas (IoT, *Internet of Things* en sus siglas en inglés) [42][43][44], de tal forma que una gran cantidad de pequeños dispositivos (medidores, dispositivos vestibles también llamados *wearables*, etc.) que producirán datos cada segundo, todo ello sin mencionar los grandes servidores y *routers* derivados del crecimiento natural de la red y los servicios web. Por datos, nos referimos a *logs* en el sentido amplio del término, en concreto, ficheros con líneas marcadas temporalmente con información interesante para los operadores en un formato conocido. Por ejemplo, a continuación se muestra una entrada de *log* del conocido servidor web Apache:

```
64.123.18.10 - - [07/Mar/2015:16:10:02 -0800] "GET /format.css HTTP/1.1"200
6291
```

En el ejemplo anterior, podemos reconocer fácilmente la marca temporal (fecha y hora), la dirección IP del servidor, el código de respuesta (200 OK) y el tiempo de respuesta (6291 microsegundos). No obstante, estas lineas deberán ser analizadas para poder extraer dichos valores.

En cuanto a la magnitud de la carga de trabajo en la captura y procesamiento de los *logs*, tanto el volumen como la tasa a la que los *logs* son producidos resultarán trascendentales. El ambicioso objetivo de esta investigación fue desarrollar un sistema capaz de manejar varios millones de líneas por segundo. Por ejemplo, si las líneas de *log* son de 300 bytes de longitud, entonces un enlace de 10 Gbps estaría totalmente saturado a partir de 3 millones de *logs* por segundo. Esta tasa sería la correspondiente a una población de 30 millones de pequeños dispositivos que generasen una línea cada diez segundos, o un millar de servidores que generasen 3,000 *logs* por segundo cada uno. Esta cantidad de datos tan vasta, superior a la que las soluciones del estado del arte soportan en el momento de esta investigación, deberá ser almacenada, filtrada, agregada y visualizada con el fin de detectar eventos o incidentes inesperados.

En lo que sigue, se describirá la motivación tras la investigación, en concreto, las necesidades que los administradores de redes y sistemas tienen durante el análisis de *logs*. Se ha ganado una experiencia vital en este ámbito a través diferentes proyectos con grandes centros de datos de bancos y grandes compañías, los cuales se han llevado a cabo con la compañía *spin-off Naudit*[4], asociada al grupo de investigación *High Performance Computing and Networking* de la Universidad Autónoma de Madrid, al cual pertenece el autor de esta tesis. Precisamente, este trabajo está motivado por muchos de los casos de uso en infraestructuras de TI, que cuentan con una cantidad masiva de *logs*, los cuales son difíciles de procesar mediante las técnicas tradicionales.

**El primer requisito es que los administradores de red deben centralizar los *logs* generados** por los sistemas actualmente en producción. Es necesario indicar que esto no es únicamente un requisito operacional, sino también legal en el sector bancario[5]. Con el propósito de enviar los *logs* desde los servidores o *routers*, Syslog [45] constituye el estándar *de facto*, ya sea a través de UDP, TCP o TLS. Por tanto, Syslog supone la mejor opción para este fin debido a su amplia implantación, la cual continuará creciendo en el futuro. En consecuencia, en el repositorio central de *logs*, un componente *Loginson* debe capturar el flujo de *logs* a la tasa objetivo de millones de líneas de *log* por segundo. Este componente provee balanceo de carga dividiendo el flujo de datos tan rápido como sea posible hacia diferentes nodos de almacenamiento y procesamiento, al estilo *Round-Robin*.

**El segundo requerimiento consiste en pre-procesar los datos en crudo y el proporcionar un resumen de los datos a alto nivel**, en el cual el pre-procesamiento implica filtrado, agregación y, en general, realizar un pre-analisis de los datos crudos. Este obstáculo tiene el handicap adicional de tener que lidiar con las características específicas de cada tipo de mensaje de *log* y su formato. Por tanto, cada tipo de *log* requiere un filtro distinto o un agregador concreto (por ejemplo, el número de códigos HTTP de error interno del servidor — 500 — cada 5 minutos). Además, la flexibilidad es un requisito clave en el componente de pre-procesamiento de *Loginson*. La salida de este componente es una versión resumida de los datos crudos, la cual es apta para realizar visualizaciones como series temporales, histogramas, ofreciendo una perspectiva general, útil para la detección de incidentes.

Por último, **el tercer requisito del sistema es la necesidad de ofrecer una representación gráfica de los datos resumidos** a través de una interfaz amigable

[4] `http://www.naudit.es`

[5] `https://www.pcisecuritystandards.org/`

que permita la visualización rápida de los incidentes. Hay que destacar que la etapa anterior de filtrado, transformación y agregación, reduce enormemente el conjunto de datos a ser visualizado, y allana el terreno hacia el uso de soluciones de código abierto (como Kibana[6]) para la creación de tableros de mando (traducción literal de *dashboard*) con unos recursos modestos (unas pocas máquinas virtuales). Por tanto, se decidió incorporar Kibana (junto con la pila de aplicaciones de Elastic) como herramienta de visualización en vez de construir desde cero un sistema propio para este fin.

### 3.1.2. Novedad

En esta investigación, se considera específicamente el caso de uso en tiempo real para la monitorización de *logs* en grandes infraestructuras de TI. Hay que destacar que la monitorización es clave para redes y centros de datos, y, en consecuencia, el caso de uso propuesto cubre la amplia demanda para la monitorización a alta velocidad. La monitorización requiere de la ingesta de *logs*, procesamiento y sistemas de visualización cuya base de datos sacrifique las funcionalidades tradicionales a cambio de velocidad, centralización y adaptación a las herramientas para la implementación de *dashboards*.

Primeramente, Syslog se ha convertido en el estándar *de facto* para el envío de *logs* en el lado del servidor. La introducción de agentes en el servidor, las cuales puedan realizar el pre-procesamiento y distribución de los logs a múltiples centros de recolección, es costosa y, en muchos casos, imposible de desplegar en servidores críticos. Este requisito impide el uso de sistemas de bases de datos distribuidas en los cuales los productores de datos envíen datos a los consumidores de datos mediante algoritmos de balanceo de carga inteligentes [46], en vez de hacer uso de un punto central para la recolección de datos. Para ser exactos, hacer que todos los servidores envíen los *logs* a un punto central es mucho más simple, ya que todos los servidores pueden hacer uso de la misma dirección IP, en concreto, aquella del centro de recolección. Este esquema no evita los requisitos de la alta disponibilidad, ya que el punto central de recolección puede ser desplegado en varios nodos. El sistema de recolección central es estático, lo cual simplifica su potencial redundancia.

En segundo lugar, las consultas realizadas por administradores de redes y sistemas serán mucho más simples que las sentencias SQL. Cuando quiera que un incidente ocurra, las consultas que típicamente son realizadas consisten en simples búsquedas de rangos de IPs y puertos dentro de un intervalo de tiempo concreto (aquel del incidente),

[6] `https://www.elastic.co/`

y no en complejas sentencias SQL. Mientras que las bases de datos clásicas proveen mejores funcionalidades para realizar la consulta, nuestro caso de uso sacrifica este tipo de características en favor de la velocidad, las capacidades analíticas [47] y mejores funcionalidades de visualización. También hay que destacar que los mensajes de *log* más recientes son los más relevantes y serán los más consultados, mientras que los antiguos serán menos y menos relevantes según transcurra el tiempo.

En tercer término, es necesario estresar la importancia de la visualización de datos en la monitorización de redes y sistemas. Con redes tan heterogéneas, con tantos servidores y redes diferentes a monitorizar, los administradores de redes y sistemas se esfuerzan en construir un sistema sencillo, pero completo, de visualización del centro de datos. Un requisito así requiere de funcionalidades de sumarización, ya que no es posible mostrar toda la información en un *dashboard*.

La novedad de *Loginson* reside en que aborda los anteriores requisitos con, además, una gran velocidad de ingestión de *logs*. Por tanto *Loginson* se diferencia de otros sistemas distribuidos más complejos que están orientados a casos de uso de recolección genérica de datos y búsqueda de datos, proporcionando un novedoso sistema *ad-hoc* para la recolección de *logs*, su procesamiento, almacenamiento y visualización.

## 3.2. Estado del arte

Gran parte de los sistemas de procesamiento y almacenamiento distribuidos son útiles para escenarios genéricos de análisis, pero específicamente para la recolección de *logs*, procesamiento y visualización. Su principal objetivo es realizar análisis complejos sobre una gran cantidad de datos persistentes [48], la cual es insertada a una tasa relativamente baja.

En realidad, los sistemas tradicionales de datos masivos (de ahora en adelante *big data*) pueden escalar horizontalmente [49] hacia un gran número de nodos, los cuales son balanceados mediante complejos algoritmos. Estos sistemas tratan de mantener oculta la complejidad de bajo nivel a los programadores, permitiendo que se centren en el problema a resolver. El sacrificio a cambio es que la abstracción obtenida añade una pérdida de rendimiento cuando se emplean pocos nodos, ya que el escalado vertical es sacrificado en favor de la simplicidad.

En el caso de *Loginson*, la pérdida de escalabilidad vertical no es aceptable. Las restricciones de tiempo real no pueden ser satisfechas mediante ninguno de los sistemas genéricos que se han evaluado, a menos que se emplee un gran número de nodos [52].

Cuadro 3.1: Resumen comparativo de los diferentes sistemas del estado del arte

| **Propósito** | **Sistema** | **Rendimiento o Características** |
|---|---|---|
| Brókers de Mensajes Brokers | Syslog-ng | 650K log/s mutli-hilo [50] |
| | Apache Kafka | 300K log/s<br>Usando un productor y un consumidor |
| | Apache Flume | 0.77M log/s/hilo |
| | Logstash | 29.5K log/s/hilo |
| | FluentD | 13K log/s/hilo |
| Bases de datos de almacenamiento | Cassandra | 251K TPS[1] en un único nodo [51] |
| | ScyllaDB | 1.8M TPS en un único nodo [51] |
| Pre-procesamiento (Dividir y unir los campos de los logs) | Apache Storm | 900K logs/s<br>Usando el 100 % de todos los núcleos |
| | AWK | 900K log/s/hilo |
| | Python | 500K log/s/hilo |
| | Perl / Ruby | 300K log/s/hilo |
| Representación Gráfica | Kibana | Herramienta oficial de Elastic, como ES[2]<br>Herramienta de diseño de dashboards<br>Fácil de usar y gráficas versátiles |
| | Grafana | No oficial, carece de soporte<br>Herramienta de diseño de dashboards<br>Limitado a series temporales[4] |
| | D3.js | Requiere un complejo *middleware*<br>Gran catálogo de visualizaciones<br>Librería JS[3], requiere desarrollo a medida |

[1] Transactions per second
[2] Elasticsearch
[3] JavaScript
[4] En el momento de esta investigación

Además, la escalabilidad vertical hace posible el uso de configuraciones activo-activo para la replicación de los datos, ya que el número de nodos involucrados no es muy alto. La conclusión alcanzada es que un sistema a medida es necesario para los componentes de análisis en tiempo real y de almacenamiento de *Loginson*. Después de reducir la cantidad de datos mediante el primero de ellos, sistemas como Elasticsearch pueden ser utilizados para ofrecer analíticas de alto nivel.

A continuación se revisará el estado del arte del caso de uso propuesto para el almacenamiento y visualización de *logs*, en el cual se concluye que ningún otro sistema, hasta la fecha, satisface los requisitos expuestos en las secciones anteriores. El estado del arte se ha dividido en cuatro secciones: bróker de mensajes y sistemas de balanceo de carga, nodos de bases de datos; componentes de pre-procesamiento (transformación de datos), y representación gráfica. A modo de resumen, la tabla 1 presenta los aspectos más destacados de las diferentes herramientas incluidas en el estado del arte.

### 3.2.1. Brókers de mensajes y balanceo de la carga para la centralización de *logs*

La elección más sencilla para la centralización de *logs* consiste en adoptar uno de los agentes de Syslog que ya existen en la comunidad de código abierto. De entre ellos, **Syslog-ng** resulta el más rápido, ya que fue diseñado precisamente con esa meta en mente. Puede ser fácilmente configurado para recibir *logs* de un agente cliente Syslog y para almacenarlos en un lugar centralizado, lo que le convierte en la solución ideal para el problema expuesto. Desgraciadamente, **Syslog-ng** no es capaz de conseguir el rendimiento de 3 millones de *logs* por segundo, alcanzando como máximo 650.000 [50].

Otra sistema popular para la centralización de *logs* es **Apache Kafka** [53], un bróker capaz de recibir información desde diversas fuentes y distribuirlos a distintos destinos. Kafka hace uso de un patrón publicación-suscripción que permite la persistencia de los datos en disco así como la distribución de ellos a los consumidores suscritos (logstash, fluentd, clientes TCP, etc.). Kafka categoriza los mensajes en temas, posibilitando que los consumidores se suscriban a uno o varios temas. Estos temas están divididos en un número de particiones, las cuales pueden ser accedidas en paralelo por múltiples consumidores. En el caso de los productores, estos pueden publicar sus mensajes de *log* en un tema específico.

En las pruebas de velocidad publicadas por Linkedin [54], un clúster formado por tres nodos fue capaz de ingerir 2 millones de eventos por segundo generados por tres productores. Las pruebas también mostraron que el rendimiento del productor no se ve reducido por el incremento de consumidores, ya que un productor puede insertar hasta 800.000 eventos por segundo con o sin consumidores.

A diferencia de Syslog-ng, Kafka no está enfocado al procesamiento de *logs* y se requiere trabajo adicional para adaptarlo a este propósito. Kafka parecía satisfacer las necesidades del caso de uso propuesto, así que se realizaron pruebas haciendo uso de un servidor con dos procesadores `Intel Xeon E5-2630 v2 @ 2.60GHz` CPUs con 32GB de memoria RAM y un sistema de almacenamiento RAID 0 formado por 10 discos duros convencionales. Se concluyó que Kafka no es capaz de satisfacer los requisitos propuestos, ya que utilizando el 100% de la CPU de la máquina, mostró una capacidad de ingesta de 300.000 entradas de *log* por segundo (con un tamaño de 177 bytes), haciendo uso de un único productor y un único consumidor.

Otros sistemas de recolección[7] de *logs* que hacen uso de Kafka también obtienen

[7] `https://www.loggly.com/`

resultados similares. Este rendimiento es un orden de magnitud por debajo de nuestros requisitos, por lo que Kafka fue descartado para su uso en *Loginson*.

Una alternativa a Apache Kafka es **Apache Flume**[8], siendo su principal diferencia respecto al anterior que éste envía la información hacia el cliente (destinatario), mientras que los clientes de Kafka consumen los datos de las colas de Kafka a su propio ritmo. Mientras que los sumideros de Flume (*sink* es el término utilizado por Flume) pueden leer datos únicamente de un canal, los datos de múltiples sumideros pueden ser leídos desde un mismo canal. Además, múltiples fuentes de datos pueden escribir en el mismo canal. La figura 3.1 compara las arquitecturas de estos dos sistemas. Al igual que los sucriptores de Kafka, los sumideros de Flume pueden ser de distintos tipos, por ejemplo, sistemas HDFS [35], Elasticsearch, HBase, etc.

Flume fue evaluado en la misma máquina con dos configuraciones distintas de sumideros: Sumidero Nulo (*Null Sink*), el cual descarta todos los eventos que recibe desde el canal [55]; y el Sumidero de Rotación de Ficheros (*File Roll Sink*) que guarda los eventos en ficheros. Para conseguir un rendimiento máximo, los ficheros de pruebas fueron guardados en un sistema de ficheros en memoria RAM. Flume consiguió un rendimiento de 1.500.000 *logs* por segundo e hilo con el primer tipo de sumidero, y cerca de 770.000 *logs* por segundo e hilo con el segundo tipo, haciendo uso de un tamaño de *log* de 291 bytes. Más detalles sobre estas pruebas pueden encontrarse en la figura 3.9 de la sección 3.4. De nuevo, esta alternativa no satisface los requisitos propuestos.

Existen otros sistemas dedicados al procesamiento de *logs* que merece la pena mencionar a pesar de que no provean el rendimiento necesario. **Logstash**[9] desarrollado por

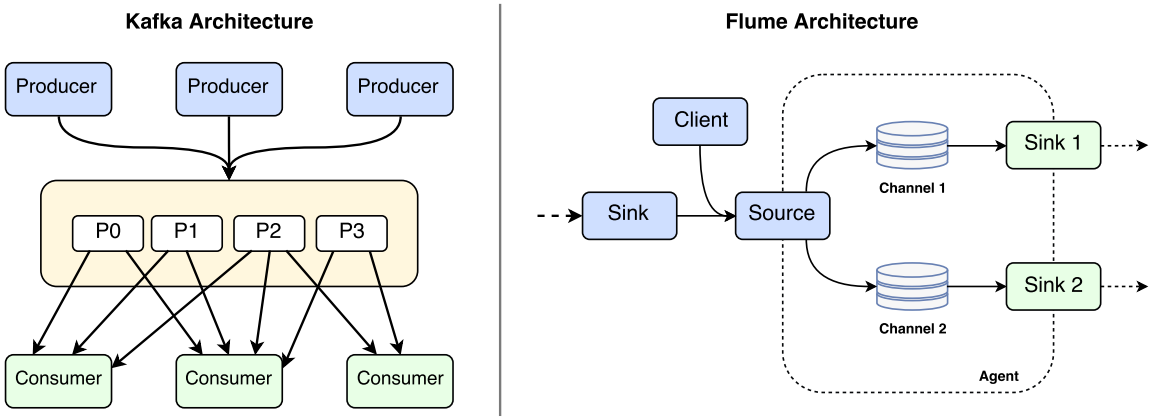



Figura 3.1: Arquitecturas de Flume y Kafka

[8] https://flume.apache.org/
[9] https://logz.io

*Elastic*, recolecta, procesa y reenvía mensajes de *log* desde y hacia diferentes sistemas (TCP, UDP, Elasticsearch, Kafka, 0mq, etc.) prometiendo un rendimiento de 50.000 eventos por segundo[10] haciendo uso de filtros *grok* para diferentes patrones de *logs*.

De forma similar, Treasure Data desarrolló **FluentD**[11], una alternativa a Logstash la cual presume de un rendimiento de 800.000 eventos por segundo [56] en su servicio de datos masivos en la nube, pero del que no especifican ninguna serie de detalles acerca del experimento realizado [57]. En la página web del producto se describe el sistema como capaz de procesar 13.000 eventos por segundo y núcleo [58].

Se realizaron pruebas con ambos sistemas, Logstash y FluentD, en la misma máquina que en los experimentos anteriores, configurando los sistemas de tal forma que no se alterase los mensajes de *logs* recibidos salvo aquellas transformaciones que sean impuestas por defecto, las cuales suelen suponer la adición de una marca temporal a los

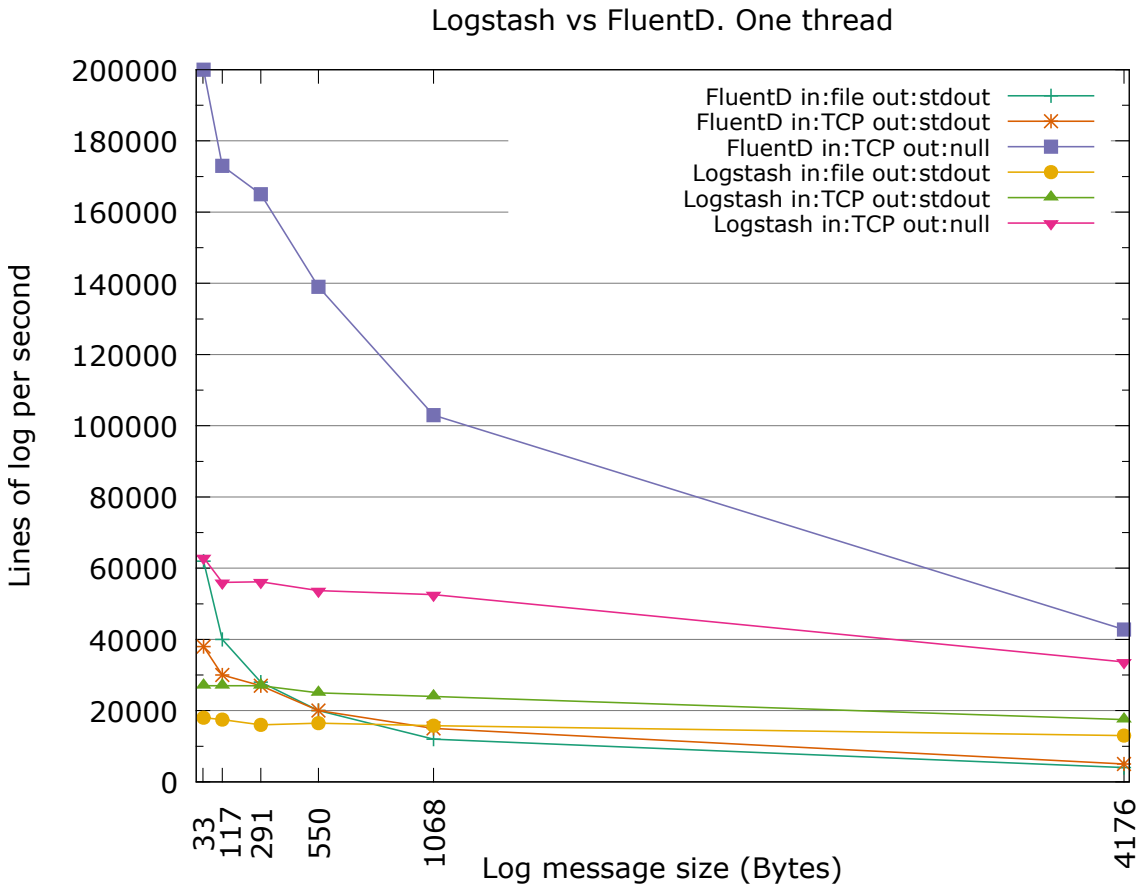



Figura 3.2: Pruebas de velocidad mediante Logstash y FluentD

[10] https://www.elastic.co/blog/logstash-1-5-0-ga-released
[11] http://www.fluentd.org/

mensajes. La primera prueba consistió en la lectura de los datos desde archivos alojados en un sistema de ficheros en RAM, mientras que en la segunda prueba se recibieron vía TCP, y en la última se recibieron de la misma manera pero fueron enviados a un destino nulo haciendo uso de sus correspondientes componentes de emisión nula.

Como puede apreciarse en la figura 3.2, ninguno de los dos sobrepasa los 60.000 *logs* por segundo excepto cuando los mensajes son enviados a un destino nulo. De acuerdo a estos resultados, estos sistemas fueron descartados.

#### 3.2.1.1. Nodos de bases de datos

Tras la recepción de los mensajes de *log* en un sistema de balanceo de carga, estos deben ser distribuidos entre varios nodos de almacenamiento y bases de datos. De esta manera el trabajo de escritura en disco es reducido, lo que deja espacio para operaciones de lectura que no afecten la velocidad de las operaciones de escritura.

Existen numerosas opciones para almacenar estos datos, desde grandes sistemas de datos masivos a simples sistemas de almacenamiento clave-valor. Las extensas pruebas realizadas mostraron que, mientras que ***Loginson* requiere de un único servidor para procesar hasta 10 millones de líneas por segundo**, otras soluciones de almacenamiento de datos masivos como **Apache HDFS** o **Casandra**[12] solo son capaces de satisfacer nuestra tasa de entrada haciendo uso de al menos 5 servidores, por lo que estas alternativas se tuvieron que desechar debido a su relación coste-capacidad [59].

Estos sistemas están muy bien adaptados a casos de uso genéricos a costa de rendimiento, ofreciendo características como tolerancia a fallos o alta disponibilidad que, a pesar de su gran valor, pueden ser conseguidas por otros medios como la redundancia completa del sistema.

En el caso de los almacenamientos de clave valor, existen numerosas pruebas de velocidad [60] que muestran que ninguno de ellos puede alcanzar una velocidad de escritura acorde a las necesidades descritas. Es más, estos sistemas están diseñados enfocados a lecturas y escrituras aleatorias. Por todos estos motivos, fueron descartados en favor de un sistema de almacenamiento secuencial en plano, ya que los *logs* son normalmente consultados en trozos secuenciales.

Algunos sistemas como **ScyllaDB**[13] [61] (una versión de Cassandra de código abierto escrito en C++) afirma conseguir hasta un millón de transacciones por segundo y servidor. Una de sus pruebas de velocidad [62] hace uso de un clúster con 3 nodos y 128GB de

[12] `http://cassandra.apache.org/`
[13] `http://www.scylladb.com/`

memoria RAM con 4 discos SSD de 960GB en RAID cada uno. Haciendo uso de un factor de replicación de 3, el clúster provee un rendimiento medio de 1,9 millones de transacciones por segundo (TPS).

Otra prueba realizada [51] usando un único nodo dio un rendimiento de 1,8 millones de TPS. Aunque el rendimiento de ScyllaDB sobrepasa con creces el de Cassandra (251.000 TPS en este último experimento), las pruebas de estos sistemas se realizaron mediante discos SSD, los cuales proporcionan un mejor rendimiento en operaciones simultáneas de escritura/lectura que los discos convencionales HDD y a un mayor coste. Por ejemplo, un sistema que recolecte 3 millones de líneas de *log* por segundo, produciría aproximadamente 42 TBytes de *logs* al día, y por lo tanto, en el momento de esta investigación, el uso de discos SSD resultaría extremadamente caro.

Otras soluciones como **MapRDB**[14] ofrecen una licencia comercial de 4.000 dólares ($) por nodo utilizado (2011) [63]. MapR consigue ingerir 100.000.000 de *puntos* por segundo mediante 4 nodos y un factor de replicación de los datos de 3 gracias a su propio sistema de ficheros mapR-FS [64].

**Splunk** es un servidor de búsquedas similar a Elasticsearch, el cual está disponible mediante licencia comercial a un precio de unos 28.750$ por 10GB/día, unos 172.500$ por 100GB/día en su versión empresarial y 15.870$ por 10GB/día en su sistema en la nube. Es un sistema de código cerrado el cual, de acuerdo a sus propios experimentos [65] [66], otorga un rendimiento de unos 80.000 eventos por segundo en una máquina Linux 64-bit HP DL380G7 con 2 procesadores de 6 núcleos Xeon @2.67 Ghz y 12GB de memoria RAM. Al contrario que Splunk, **Elasticsearch** es una solución de código abierto que obtiene el mismo rendimiento con un procesador dual Xeon X2699 @2.3 Ghz y 256GB de memoria RAM haciendo uso de 4GB de memoria de pila asignada, de acuerdo a sus propias pruebas de velocidad [67].

### 3.2.2. Pre-procesamiento de los *logs*

Para el pre-procesamiento de los datos en crudo, un sistema de transmisión continua (*streaming*) en tiempo real resulta necesario. Una solución popular en la literatura es **Apache Storm**[15], un sistema de tiempo real para computación distribuida. Se realizaron pruebas de velocidad mediante un nodo con dos procesadores de 6 núcleos Intel Xeon E5-2630 v2. Utilizando el 100 % de todos los núcleos, Storm es capaz de transferir 900.000 *logs* por segundo, tan sólo realizando una separación y unión de los valores del mensaje.

[14] https://www.mapr.com/

[15] `http://storm.apache.org/`

Otros sistemas como **Apache Spark**[16] streaming o **Apache Flink**[17] obtuvieron aún peor rendimiento. El resultado de 900.000 *logs* por segundo mostrado en la tabla fue obtenido mediante el lenguaje AWK, haciendo uso de un único núcleo. Este rendimiento se reduce a unos 500.000 *logs* por segundo haciendo uso de Python y a 300.000 *logs* por segundo mediante Ruby o Perl. Lenguajes como AWK o Python pueden ser enlazados haciendo uso de pequeños módulos de envoltorio o tuberías UNIX para crear una línea de procesamiento continuo (*streaming pipeline*). Estas soluciones tienen el beneficio extra de ser fáciles de utilizar y bien conocidas, sin necesidad de programadores especializados en el área.

#### 3.2.2.1. Serialización

El problema de la serialización de los datos surge cuando se requiere el procesamiento a altas velocidades para convertir los mensajes de *log* de su formato original al formato *JavaScript Object Notation* (JSON). Uno de las principales limitaciones con las que se trabajan es el obstáculo de no poder incorporar ningún software en los emisores de mensajes ya que normalmente son sistemas en producción que no deben ser alterados. En consecuencia, el formato de los mensajes no puede ser alterado en la fuente de los datos a pesar de que herramientas como Apache, Nginx o incluso Syslog-ng OSE (el cual es capaz de enviar mensajes directamente a Elasticsearch) [68] soporten la creación de mensajes en formato JSON.

Como se ha explicado previamente, sistemas como Logstash o FluentD no son suficientemente rápidos para realizar la recolección o el procesamiento de los mensajes. La velocidad de conversión está entre 15.000 y 30.000 mensajes de *log* dependiendo del tamaño de estos.

### 3.2.3. Representación gráfica

El sistema de visualización es una parte esencial de la arquitectura ya que es con la cual va a interaccionar el usuario final del sistema, y por lo tanto, la simplicidad y flexibilidad son fundamentales para mejorar la experiencia del usuario. En este ámbito, las gráficas deben ser suficientemente dinámicas y flexibles para satisfacer las necesidades de los analistas de datos, y aun así, suficientemente rápidas para obtener una vista general en cuestión de segundos.

[16] `http://spark.apache.org/`
[17] `https://flink.apache.org/`

Algunas herramientas como **D3.js**[18] (una librería JavaScript) poseen un gran catálogo de distintos tipos de gráficos (mapas de árboles, gráficos de cajas, de bigotes, burbujas, etc.) pero requieren de un desarrollador web con buenas habilidades para hacer cualquier cambio en estas visualizaciones, además del hecho de que requieren una capa intermedia entre las gráficas y la base de datos, proveyendo la información que va a ser visualizada.

Más aún, herramientas como **Grafana**[19] o **Kibana** son capaces de consultar fuentes de datos y mostrar la información mediante una gran variedad de gráficas sin necesidad de habilidades programativas. En el casos de Grafana, en el momento de esta investigación, no posee gran variedad de visualizaciones y está enfocada a series temporales.

Finalmente, Kibana tiene tanto variedad de gráficas (tartas, series temporales, histogramas, etc.) y una API para conectarse a índices de Elasticsearch de donde consultar la información. Esto hace que sea realmente sencillo modificar los *dashboards*. Además, Kibana y Elasticsearch están en constante desarrollo por el equipo de Elastic, el cual, recientemente presentó un componente llamado **Timelion** con funcionalidades enfocadas a series temporales (derivadas, medias móviles, etc.). Además, Kibana también soporta la generación de *dashboards* embedidos permitiendo la creación de webs personalizadas.

Todas las características mencionadas anteriormente hacen de Kibana la solución elegida para la visualización de los datos en *Loginson*.

## 3.3. La arquitectura de *Loginson*

En *Loginson* los mensajes de *logs* son recibidos en un servidor centralizado llamado *LogFeeder* que añade información como las marcas temporales de los mensajes de *logs* con el fin de evitar problemas con datos provenientes de distintos husos horarios o de fuentes de datos mal sincronizadas. Después, son distribuidos entre varios nodos de almacenamiento, utilizando el método *Round-Robin*. El almacenamiento consiste en ficheros planos de 1GB de tamaño. Este tamaño es suficientemente largo como para que aproveche las ventajas en escritura y lectura ofrecidas por las operaciones secuenciales de los discos de aguja. Si se requiere de alta disponibilidad, un sistema *Loginson* activo-activo con replicación de datos puede ser utilizado para evitar la pérdida de datos. Un sistema de replicación de datos así no resultaría caro debido a la alta utilización de los nodos (escalado vertical), en contraste con la baja utilización de los sistemas de escalado puramente horizontal.

[18] `https://d3js.org`
[19] `http://grafana.org`

Es necesario mencionar que el sistema propuesto provee un alto rendimiento gracias a su simplicidad y capacidad de escalado vertical. Si se requiriera mayor rendimiento, se podrían añadir más nodos de almacenamiento para dividir la carga de trabajo. El punto central de recolección *LogFeeder* resulta el cuello de botella del sistema, sin embargo, puede ser aliviado mediante el uso de un sistema de balanceo de carga previo para distribuir la carga hacia múltiples nodos de *LogFeeder*. En cualquier caso, el rendimiento de *LogFeeder* alcanza varios millones de mensajes de *log* por segundo (ver figura 3.6), el cual es suficiente para satisfacer las necesidades de grandes infraestructuras de TI.

Mientras que los *logs* son almacenados en los nodos de bases de datos, un resumen es construido haciendo uso del lenguaje AWK (por ejemplo, medias móviles de los datos), y enviados a un componente llamado *LoginsonReceptor*. Este componente centraliza el procesamiento de los *logs* y los inserta en una base de datos de Elasticsearch, de tal forma que Kibana pueda mostrar visualizaciones de las diferentes métricas del resumen de los datos.

Normalmente, *LoginsonReceptor* podrá ser alojado en un nodo diferente, de tal forma que el acceso a la información resumida no interfiera con la recolección de nuevos datos. Para realizar una consulta en profundidad de los datos, un API web permite acceder a los *logs* almacenados en los nodos de bases de datos. Este componente llamado *DBController* es capaz de encolar varias operaciones de lectura, las cuales pueden ser abortadas en cualquier momento.

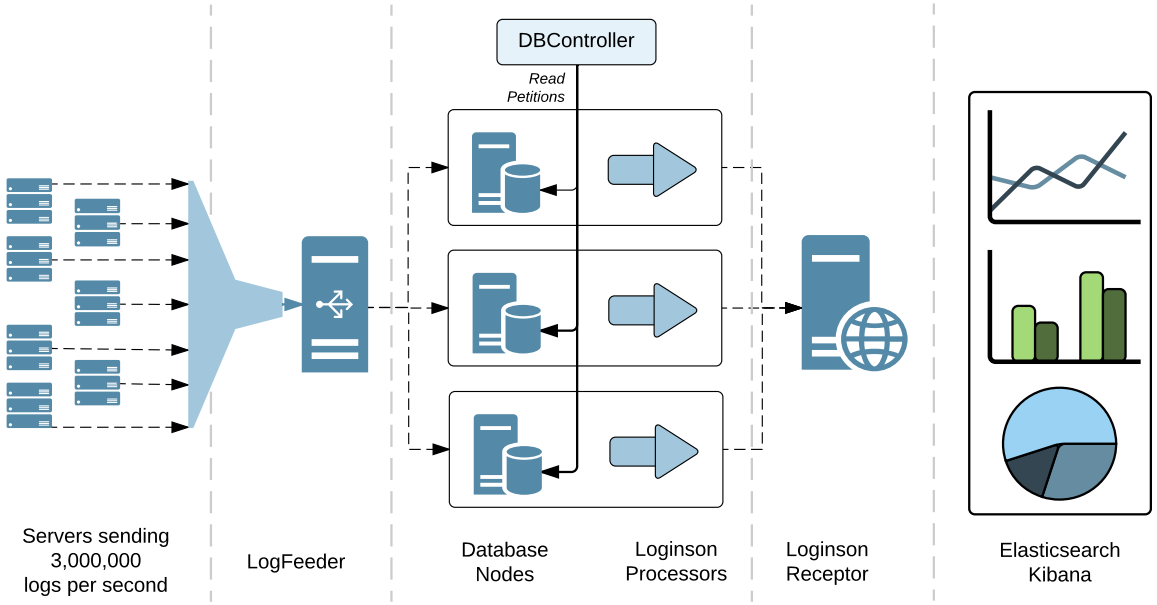



Figura 3.3: Arquitectura de *Loginson*

Por tanto, el usuario es capaz de abortar o modificar las peticiones de lectura sin esperar a que finalicen. Los mensajes son entonces procesados de la misma manera que en el modo en tiempo real y enviados a *LoginsonReceptor* para su indexación en Elasticsearch.

En lo que sigue, se describirá en detalle cada uno de los componentes anteriormente mencionados. La figura 5.2 muestra la arquitectura de *Loginson*.

### 3.3.1. Brókers de mensajes y balanceo de la carga para la centralización de *logs*

Se optó por realizar un balanceo de carga desde el nodo central a los nodos de las bases de datos mediante el propio sistema en vez de hacer que los clientes enviasen los *logs* directamente a los nodos específicos de bases de datos. Así, se evita que algunos nodos de bases de datos reciban más carga de trabajo que otros y se simplifica la configuración en los clientes. Este nodo de balanceo de carga es llamado *LogFeeder*, el cual recibe *logs* a través de UDP haciendo uso del controlador de alto rendimiento HPCAP [31] para tarjetas de red Intel® Ethernet 10 Gb PCI Express, capaz de recibir tráfico a 10 Gbps.

Para conseguir altas velocidades, *LogFeeder*, como se aprecia en la figura 3.4, hace uso de múltiples hilos que comparten información mediante una cola circular de memorias intermedias (*buffers*). Esta cola está protegida mediante un semáforo (*mutex*). Las memorias intermedias se utilizan para evitar bloquear el semáforo para cada mensaje, lo cual reduciría gravemente el rendimiento.

Uno de los hilos, llamado *Receiver Thread*, lee los mensajes desde la tarjeta de red, les añade una marca temporal y los escribe en una memoria intermedia en la cola circular. Esta memoria intermedia se vuelve disponible para su procesamiento, y el Receiver Thread selecciona la siguiente memoria intermedia en la cola para escribir los siguientes mensajes. El resto de hilos, llamados *Header Threads*, leen las memorias intermedias que están llenas con una prioridad de tipo *el que llega primero es atendido primero* (*First-come, first-serve*). Para cada *log* en la memoria intermedia, estos hilos añaden una cabecera con información del origen del mensaje y lo envían a los nodos de bases de datos usando el método *Round-Robin*. La razón para usar *Round-Robin* sobre otros algoritmos más complejos es que, de forma sencilla y eficiente, los mensajes son uniformemente distribuidos entre los nodos de bases de datos. Provocando además, que, aprovechando el paralelismo, los subsiguientes accesos a la base de datos sean más rápidos.

### 3.3.2. Nodos de bases de datos

Los mensajes de *log* son ordenados mediante la marca temporal y serán accedidos en tandas para su análisis. La mayoría de las bases de datos existentes no son adecuadas para este tipo de tarea, ya que están optimizadas para escrituras y lecturas aleatorias. En vez de esto, cada nodo de almacenamiento trabaja con ficheros planos secuenciales. Dentro de estos ficheros, los mensajes están precedidos de una cabecera de 64 bytes en la cual se incluye la marca temporal y el tipo de mensaje del que se trata.

#### 3.3.2.1. Escritura en la base de datos

La recepción de los mensajes de log se realiza mediante TCP desde el *LogFeeder* y son copiados a uno de las múltiplas memorias intermedias de la piscina de estructuras. Cuando una de estas memorias intermedias está llena, se escribe a disco de forma asíncrona como un único fichero. De esta manera la base de datos no se bloquea con esta tarea y continúa recibiendo mensajes de *log* y copiándolos en otra memoria intermedia. Como se mencionó anteriormente, el tamaño de estas memorias intermedias es de 1 GB para aprovechar las rápidas operaciones secuenciales de los discos de agujas. Estos ficheros son entonces indexados en una base de datos SQL en la cual se guarda la primera y la ultima marca temporal de los datos. Este ultimo índice será utilizado para obtener una lista de ficheros cuando estos necesiten ser consultados en un cierto intervalo de tiempo. Mientras que este hilo está recibiendo mensajes de *log*, otro está procesándolos tal como se explicará en la sección 3.3.3.

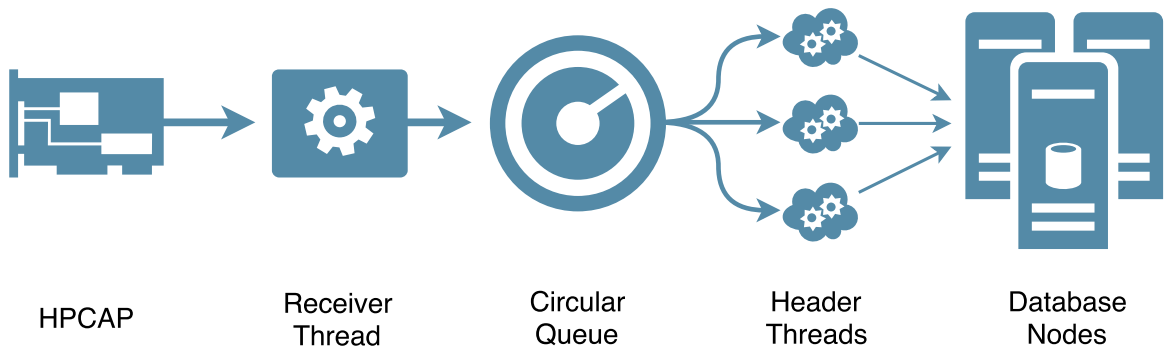



Figura 3.4: Arquitectura de *LogFeeder*

#### 3.3.2.2. Lectura de los *logs*

Los mensajes de *log* son distribuidos entre varios nodos, de tal forma que resulta necesario un único punto de acceso, llamado *DBController*. Este recibe las peticiones de lectura y las reenvía a cada uno de los nodos de bases de datos. Las peticiones de lectura se realizan mediante un API HTTP y son encoladas hasta que los nodos de bases de datos están disponibles para lectura. Es necesario indicar que estas consultas pueden ser abortadas incluso aunque estén en proceso de resolución.

Cuando un nodo de base de datos recibe una petición de lectura desde el *DBController*, este primero busca en la base de datos SQL aquellos ficheros que contengan *logs* en el intervalo de tiempo consultado a partir de las marcas de tiempo almacenadas. Cada fichero es cargado a memoria y sólo los mensajes del tipo solicitado son procesados y enviados a *LoginsonReceptor* para su posterior visualización.

### 3.3.3. Pre-procesamiento de los *logs*

Para proporcionar una vista general de los mensajes de *log* en tiempo real, se genera un resumen de determinadas analíticas (v. gr., la media móvil del tiempo de respuesta HTTP) de cada tipo de mensaje, el cual es enviado a su correspondiente proceso AWK a través de una tubería UNIX. Cada uno de los nodos de bases de datos utiliza estos *scripts* AWK, uno por cada tipo de *log*, los cuales son absolutamente flexibles, permitiendo realizar filtrados, agregaciones, o transformaciones de los mensajes así como su serialización en formato JSON, el cual es estrictamente necesario para trabajar con Elasticsearch.

El resultado es entonces enviado vía TCP a *LoginsonReceptor* (ver figura 3.5) en el servidor de visualización de datos, donde los datos de cada base de datos son recolectados e indexados en Elasticsearch para su posteior visualización mediante Kibana o Timelion. *LoginsonReceptor* se encarga de la creación de índices y *mappings* en Elasticsearch para cada tipo de mensaje de *log* haciendo uso del API Java de Elasticsearch. AWK fue elegido por su flexibilidad y velocidad tal como se muestra en la sección 3.4.3, así como por sus capacidades de alto nivel como el uso de matrices asociativas o expresiones regulares.

En la siguiente sección se expone la evaluación del rendimiento de cada uno de los componentes de la arquitectura así como del sistema completo.

## 3.4. Evaluación del rendimiento de *Loginson*

En esta sección se exponen los extensos experimentos de estrés a los que se ha sometido a *Loginson* para asegurar que satisface los requisitos de rendimiento expuestos. Se probado cada componente de forma independiente y, después, se ha evaluado el sistema completo.

En primer lugar, se ha estudiado el rendimiento de *LogFeeder* en función del número de hilos utilizados. También se ha comparado la entrada de datos con la salida de datos y comprobado si estos datos eran iguales. Después, se han probado los nodos de bases de datos y realizado pequeñas configuraciones con el fin de conseguir una velocidad de escritura constante en discos de aguja convencionales. Además, se ha verificado que el pre-procesado de los mensajes no afecta a la velocidad de escritura. De nuevo, las pruebas mostraron que la información escrita en disco era exactamente la misma que la recibida en *LogFeeder*.

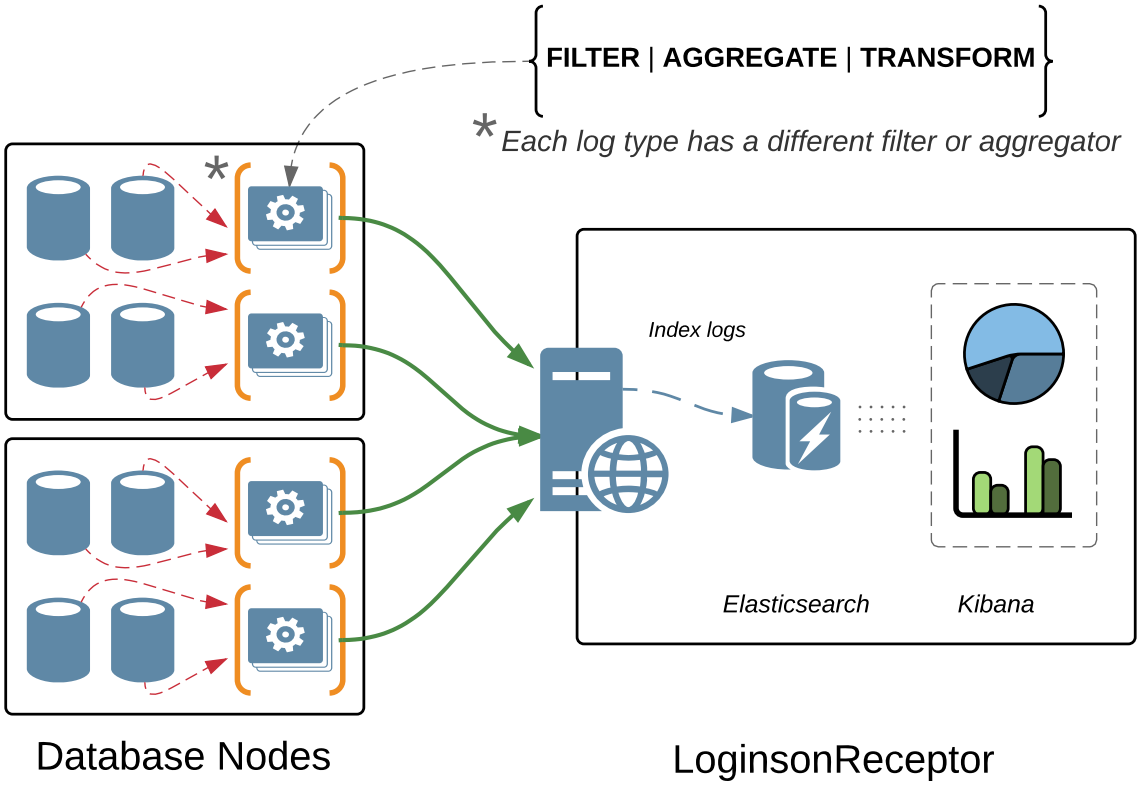



Figura 3.5: LoginsonReceptor

Cuadro 3.2: Rendimiento de *LogFeeder* con *logs* de 291 Bytes.
Cantidad de *logs* que pasan por el sistema cada 100 ms.

| Number of Header threads | Mean | Median | Standard Deviation |
|---|---|---|---|
| 1 | 512600 | 516100 | 17945.03 |
| 2 | 693200 | 701900 | 25254 |
| 3 | 1038000 | 1044000 | 17588.47 |
| 4 | 1070000 | 1073000 | 30756.99 |

Después de todo esto, para la etapa de pre-procesamiento, se compararon diferentes opciones para la serialización de los mensajes, ya que es un paso obligatorio antes del reenvío de los mensajes a *LoginsonReceptor*. Estas tres opciones resultaron más rápidas que los resultados obtenidos mediante Logstash o FluentD (ver figura 3.2) y se optó por utilizar AWK debido a su velocidad y flexibilidad. También se evaluó la velocidad de *LoginsonReceptor* indexando mensajes en Elasticsearch con el fin de comprobar la velocidad máxima de escritura de este último.

Finalmente, se realizaron dos pruebas del sistema completo. Con este propósito, se utilizaron cuatro nodos, uno de ellos ejecutando *LogFeeder*, dos como bases de datos y otro ejecutando *DBController* y Elasticsearch. En un último experimento sólo un nodo con un RAID 0 fue utilizado para almacenar *LogFeeder* y la base de datos. Es importante y necesario destacar que Flume es el único sistema que se acerca a nuestros requisitos, y por tanto, esta sección concluye con una comparativa de rendimiento entre Flume y *Loginson*, desechando el resto de alternativas.

### 3.4.1. *LogFeeder*: Brókers de mensajes y balanceo de la carga para la centralización de *logs*

La evaluación de *LogFeeder* se realizó mediante un servidor con dos Intel Xeon E5-2630 v2 @ 2.60 GHz con 32GB de memoria RAM divididos uniformemente entre los dos nodos NUMA. Cada CPU incluye 6 núcleos con el sistema *hyperthreading* desactivado. El servidor estaba ejecutando el sistema operativo CentOS 7, el cual fue configurado mediante *taskset* para garantizar que solamente la primera CPU era utilizada.

En el primer test, se hizo uso de *LogFeeder* para centralizar mensajes de *log* de diferentes tamaños entre 33 y 4176 bytes. El rendimiento obtenido se muestra en las figuras 3.6 y 3.7, así como en la tabla 3.2. Se advierte que para cualquier tamaño de mensaje, *LogFeeder* proporciona un rendimiento superior a 10 Gbps. También es capaz

de procesar más de 10 millones de mensajes de *log* por segundo, si la longitud de estas es menor que 291 bytes, lo que, como muestra la figura 3.8, resulta lo más común y probable.

Con el fin de cuantificar el tamaño típico de los mensajes de *log*, se estudió una muestra de mensajes obtenidos de un sistema real en producción. Se tomaron varios días de *logs* de Apache y se representó la función de distribución acumulada inversa (*Complementary cumulative distribution function*, CCDF) del tamaño de *log* tal como se aprecia en la figura 3.8. Se destaca que el tamaño mediano de los mensajes de *log* es de 117 bytes y que el 99% de los mensajes son más pequeños que 291 bytes, lo cual refuerza el punto de que un sistema de recolección de *logs* debe ser capaz de trabajar con tamaños de mensaje pequeños a alta velocidad.

También se observa que el mayor rendimiento se alcanza cuando se hace uso de entre tres y cuatro hilos de procesamiento, dependiendo del tamaño del mensaje. En la figura 3.9 se compara Apache Flume contra el rendimiento dado por *Loginson*.

Dicha figura muestra el rendimiento de los dos sistemas haciendo uso de 2 hilos, uno para el productor (al que hemos llamado *Receiver Thread*) y otro para el consumidor (*Header Thread*). Como puede observarse, *Loginson* supera a Flume con tamaños de *log* pequeños, los cuales son los más comunes, tal y como se ha visto en la CCDF anterior.

El segundo experimento evalua la estabilidad del rendimiento de este componente. Con este fin, se ejecutó *LogFeeder* durante más de una hora, midiendo el número de *logs* que pasaban por el sistema cada 100 milisegundos. En la figura 3.10 se muestran los resultados, observándose que el rendimiento permanece constante en el tiempo.

### 3.4.2. Nodos de bases de datos

Para la evaluación de los nodos de bases de datos se hizo uso de un servidor con 64 GB de memoria RAM, dos procesadores Intel Xeon E5-2620 v3 @ 2.40 GHz con seis núcleos cada uno, y un sistema de almacenamiento de varios discos de 7200 RPM. Los procesos de escritura fueron ejecutados siempre con el comando *ionice*, especificando la prioridad de tiempo real, la cual es la máxima posible.

El primer objetivo de este experimento es comprobar si el proceso de la base de datos es capaz de aprovechar al máximo las capacidades de lectura y escritura de los discos duros. Con este fin, primero se escribieron ficheros de 1 GB directamente desde memoria, determinando que la máxima velocidad posible en el disco duro es de 170 MB/s. Después, se ejecutó un proceso el cual recibía mensajes de *log* desde *LogFeeder*, resultando en una velocidad de escritura algo menor de unos 160 MB/s (ver figura 3.11), debida principalmente al pre-procesamiento adicional.

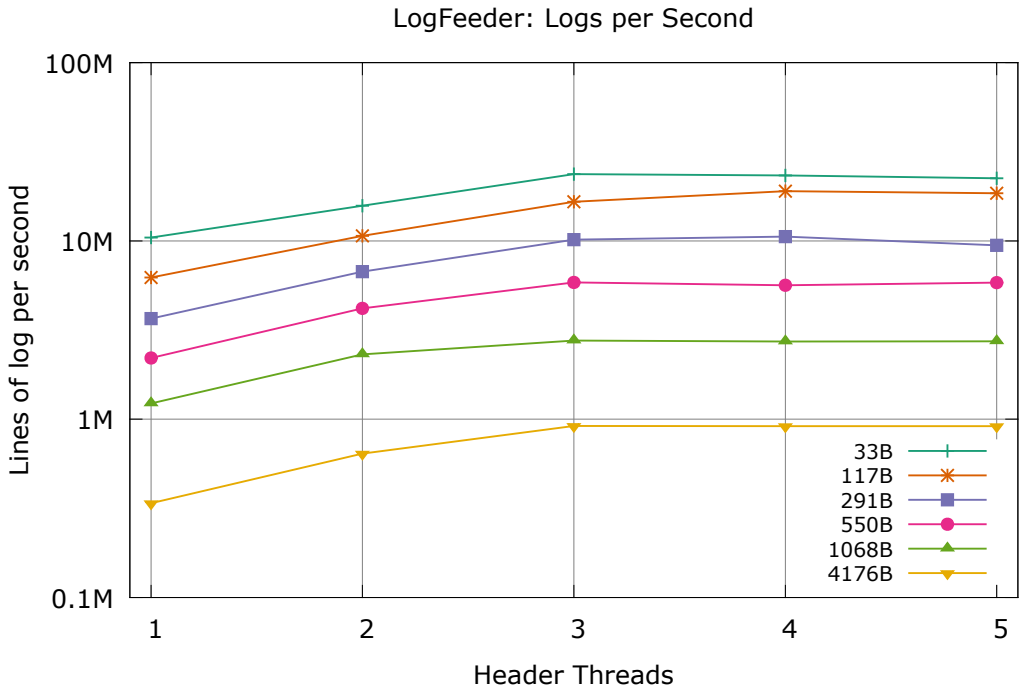



Figura 3.6: Velocidad de LogFeeder en *logs* por segundo en función del número de hilos.

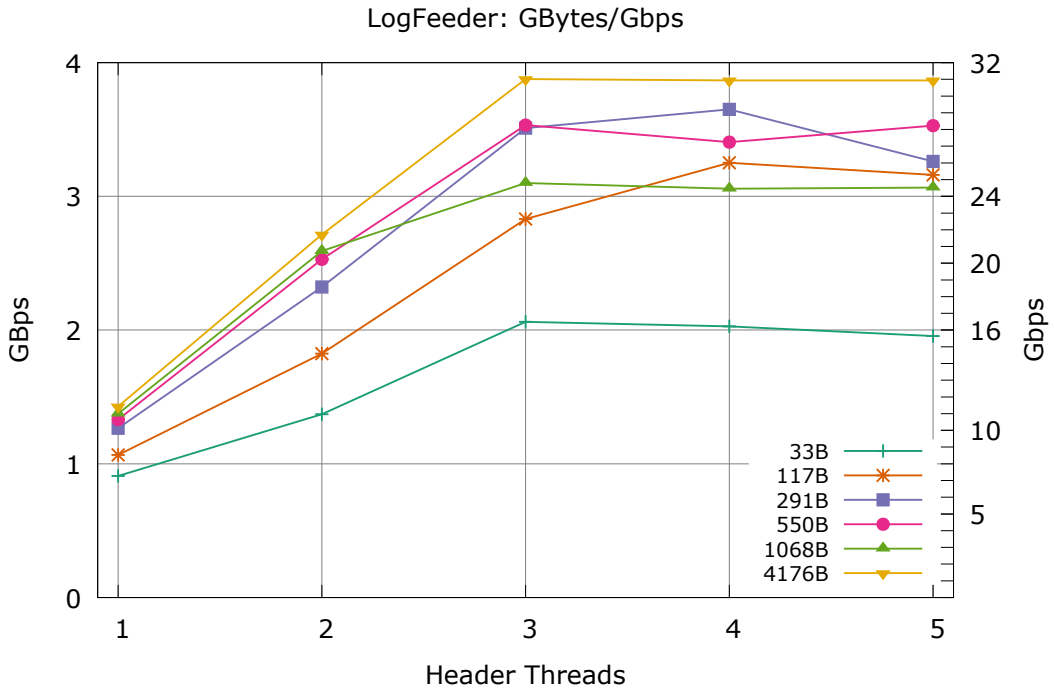



Figura 3.7: Velocidad de LogFeeder en Gbps y GBps en función del número de hilos.

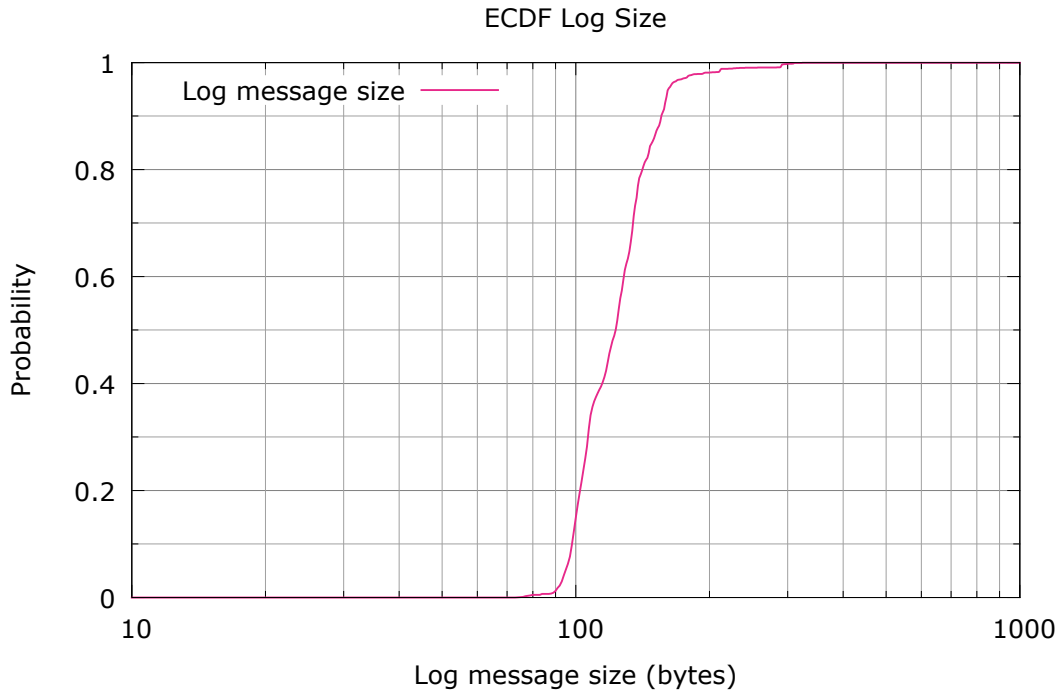



Figura 3.8: CCDF del tamaño de mensaje de *log*

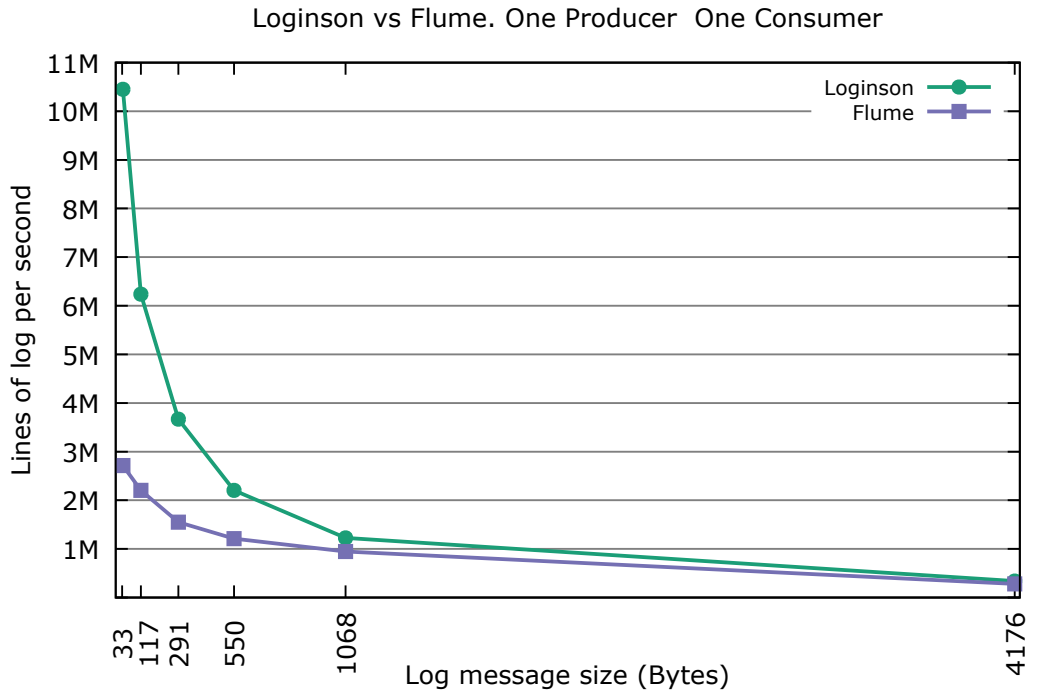



Figura 3.9: Comparación del rendimiento de Loginson versus Flume

Tras esto, se propuso encontrar la velocidad máxima de escritura a ser usada de tal forma que las operaciones de lectura no afectasen al rendimiento de la escritura, evitando pérdidas. La prioridad de los procesos de lectura fue establecida a *ocioso* mediante el comando *ionice*. Esta prioridad establece que las operaciones de entrada/salida (E/S) sean únicamente ejecutadas si no hay ningún otro proceso accediendo al disco.

Consecuentemente, se inició el proceso de la base de datos limitando la velocidad de escritura al 25%, 50%, 75% y 100% de la velocidad máxima de escritura, lo cual corresponde a 40 MB/s, 80 MB/s, 120 MB/s y 160 MB/s respectivamente. Una vez el proceso de escritura está estabilizado, se ejecutó una operación de lectura larga, mientras se continuaba midiendo ambas velocidades de lectura/escritura. También se comprobó un caso de uso en el cual se leyese de la base de datos sin estar realizando operaciones de escritura al mismo tiempo. Estos resultados se muestran en la figura 3.11.

Es necesario destacar que hacer uso de más de un 50% (80 MB/s) de la velocidad de escritura máxima reduce el rendimiento del proceso de escritura en el caso de realizar una larga operación de lectura simultáneamente. Gracias a estos experimentos se pudo determinar que el número de discos duros necesarios para procesar 3 millones de lineas

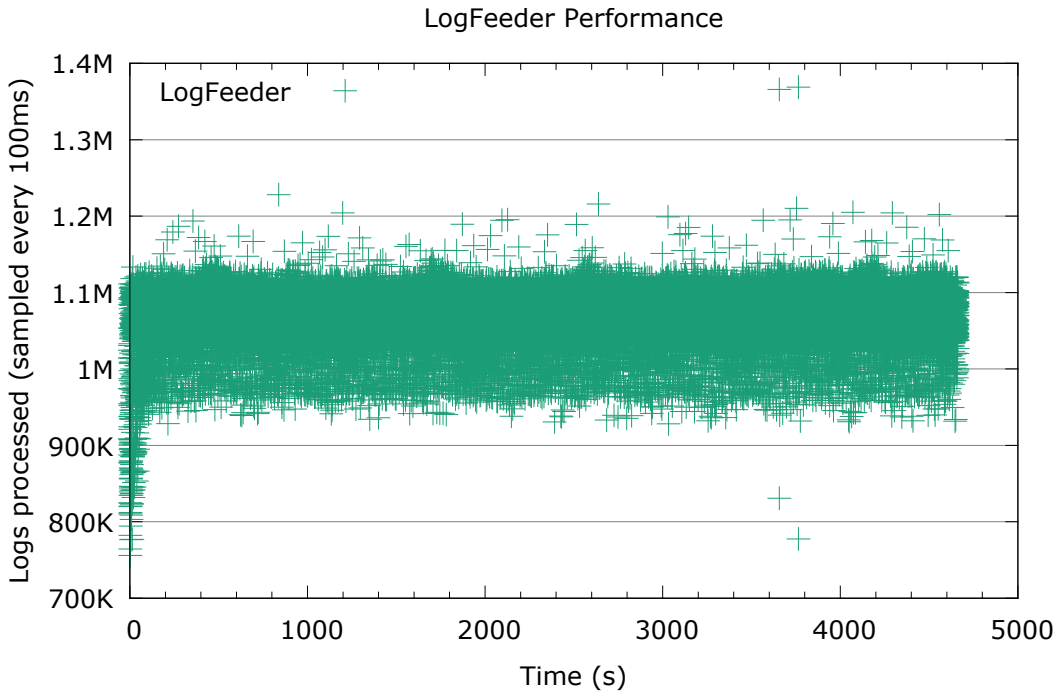



Figura 3.10: Rendimiento de *LogFeeder* usando 4 *Header Threads* con *logs* de 219 bytes

de *log* por segundo es igual a 7 discos cuando se hace uso de mensajes de 117 bytes, y 14 discos en el caso de longitudes de 291 bytes, añadiendo a estos valores los 64 bytes de la cabecera usados en la base de datos. Las longitudes de 117 y 291 son, respectivamente, la mediana y el 99º percentil del tamaño de *log* vistas en la figura 3.8.

### 3.4.3. Pre-procesamiento de los *logs*

Como se ha explicado en la sección anterior, los *scripts* de pre-procesamiento son completamente maleables, y por tanto, el rendimiento de cada *script* depende del proceso realizado para cada tipo de mensaje, ya sea filtrado, agregaciones, y/o transformaciones de los mensajes. No obstante, se realizaron pruebas con distintas métodos de serialización de mensajes, (ya que es el único paso obligatorio durante el pre-procesamiento antes de enviar los mensajes a *LoginsonReceptor*), con distintos *scripts*, enviando un fichero de *logs* de Apache con 1 millón de líneas, cada una de 100 bytes, todo ello midiendo cuánto tiempo tomaba a cada *script* la serialización del corpus. Se eligieron diferentes lenguajes

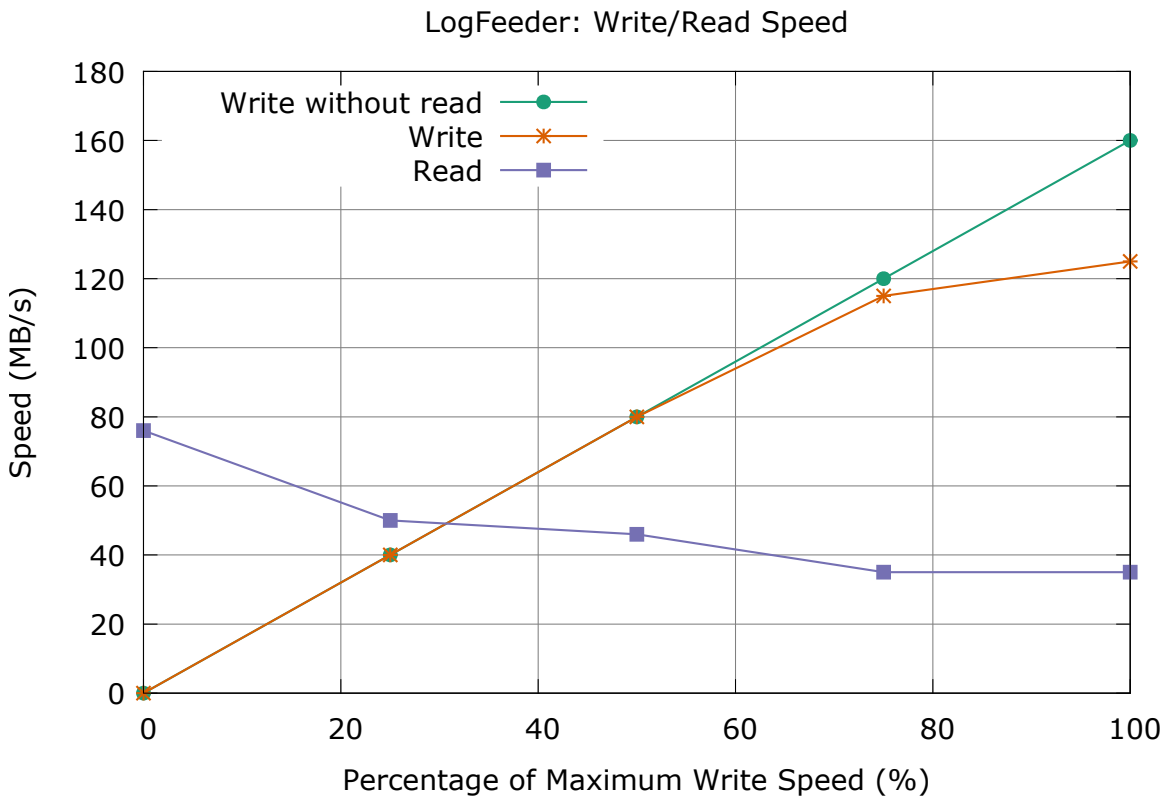



Figura 3.11: Velocidad de Escritura/Lectura de LogFeeder

como Flex, AWK y C, teniendo en cuenta que solamente necesitamos procesar alrededor de 3 millones de líneas de *log* por segundo debido al procesamiento distribuido de cada tipo de *log* con su correspondiente *script* AWK.

Es necesario indicar que Flex y Bison, un analizador léxico y un generador de procesadores sintácticos respectivamente, fueron usados para definir los patrones de cada mensaje. Bison define la estructura del mensaje a ser analizado y Flex la expresión regular de cada campo del mensaje. Estas herramientas son usadas habitualmente para definir semánticas y desarrollar compiladores usando patrones y expresiones regulares. Su mayor atractivo es que permite definir un conocimiento previo del mensaje mediante reglas, patrones y sub-patrones con diferentes prioridades, siendo capaz de manejar campos opcionales de una manera muy versátil. Después, este código puede ser compilado con *g++*. El procesador resultante es robusto pero complejo debido al sistema de búsqueda de patrones que realiza a partir de las reglas. El uso de estas herramientas requiere un desarrollador entrenado y con conocimientos.

Un *script* mucho más sencillo fue realizado mediante el lenguaje AWK, el cual es bastante más fácil de usar que Flex y Bison. Un programa AWK está compuesto de una secuencia de sentencias patrón-acción. Cada mensaje es troceado en una serie de campos en base a un delimitador. AWK es también capaz de buscar expresiones regulares en las cadenas de texto. Esta flexibilidad y facilidad de uso son grandes atractivos que hacen AWK un buen candidato, ya que el código es mucho más fácil de entender y modificar. El *script* del experimento simplemente rompe el mensaje en base al delimitador establecido, imprime el mensaje en formato JSON, y añade los nombres de los campos. Este código es interpretado sin necesidad de compilación. AWK también dispone de funcionalidades de alto nivel que lo hacen ideal para el resto de operaciones de pre-procesamiento como las de transformación, filtrado o agregaciones, gracias a su soporte para matrices asociativas y expresiones regulares.

Finalmente, se utilizó C como tercera opción, con un programa *ad-hoc* en el cual los campos fueron asignados a una estructura y más adelante impresos en formato JSON, añadiendo también los nombres de los campos. Aunque el código hace lo mismo que el *script* AWK, la complejidad es mucho mayor y el código requiere una compilación previa al igual que con Flex. También requiere una programación más elaborada.

Para los dos últimos programas, es necesario un ejercicio extra de programación si se quieren considerar campos interdependientes y/o opcionales en el mensaje a procesar (si los hubiera) pero, a pesar de esto, el código es mucho más sencillo de manejar para el programador de sistemas, comparado con las sintaxis de Flex y Bison.

Cuadro 3.3: Pruebas de velocidad para la serialización

| Sistema | Duración (segundos) | Líneas por segundo |
|---|---|---|
| Flex | 13.9 | 71,942 |
| AWK | 3.5 | 285,714 |
| C | 1.27 | 787,401 |

Como se muestra en la tabla 3.3, las tres opciones son más rápidas que Logstash (29.5 KLogs/s/hilo) y FluentD (13 KLogs/s/hilo) (ver también la figura 3.2) en la conversión de mensajes de *log* a formato JSON. De entre las opciones evaluadas, Flex es el más lento debido a su complejidad en las reglas usadas para realizar búsqueda de patrones.

La solución en C es 10 veces más rápida que Flex, pero, a pesar de esto, AWK supone un buen compromiso entre flexibilidad y velocidad, siendo 4 veces más rápido que el procesador léxico pero mucho más fácil de implementar y dominar que las alternativas.

#### 3.4.3.1. *LoginsonReceptor*

Con el fin de evaluar *LoginsonReceptor* y Elasticsearch, se hizo uso de una máquina Intel i7 @ 2.4 Gzh, con 16 GB de memoria RAM y un disco SSD PCI-e. Un total de 30 millones de líneas previamente serializadas fueron enviadas a *LoginsonReceptor* en un único nodo de Elasticsearch, las cuales fueron indexadas a una velocidad media de 83 Klineas/s, siendo Elasticsearch el cuello de botella de este experimento. Esta velocidad de ingesta de mensajes es 30 veces menor que los 3 millones de lineas por segundo recibidos al inicio del sistema, pero se considera que el rendimiento de este componente es suficiente para la realización de un resumen en tiempo real de las estadísticas analizadas, el cual puede ser utilizado para la detección de errores, anomalías o incidentes en el sistema, haciendo uso de los mensajes relevantes extraídos en etapas anteriores. Un ejemplo de esto podría ser las transacciones HTTP, las cuales podrían ser filtradas en la fase de pre-procesamiento de tal forma que sólo peticiones lentas o aquellas con errores fuesen enviadas a *LoginsonReceptor*.

### 3.4.4. Sistema completo

La evaluación del sistema completo se hizo haciendo uso de 4 servidores. *LogFeeder* fue desplegado en la misma máquina que en el experimento independiente de la sección 3.4.1, y de forma paralela, 13 procesos de bases de datos fueron ejecutados (uno por cada disco duro) en dos servidores como los utilizados en la evaluación de la velocidad

de almacenamiento en la sección 3.4.2. Un cuarto servidor fue utilizado para hospedar Elasticsearch y el *BDController*.

Para imitar los requisitos establecidos, *LogFeeder* recibió 3 millones de mensajes de *log* por segundo y los distribuyó entre los procesos de bases de datos, los cuales escribieron en los discos duros al 50 % de su capacidad de escritura. Mientras *LogFeeder* insertaba *logs* en los nodos de bases de datos, se inició una operación de lectura de larga duración sobre los datos previamente guardados. Con esta prueba se confirma que las operaciones de lectura no interfieren en las de escritura. Los *logs* fueron procesados y recibidos satisfactoriamente en Elasticsearch, siendo visualizados posteriormente en Kibana. La figura 3.12 muestra como el experimento fue realizado.

Para la ultima prueba se ejecutó *LogFeeder* y un nodo de base de datos en el mismo servidor utilizado en la prueba independiente descrita en la sección 3.4.1. Este servidor emplea un sistema de 10 discos en RAID 0 capaz de almacenar datos a 10 Gbps. **Se confirmó que el sistema *Loginson* era capaz de recibir *logs*, almacenarlos en el sistema RAID y procesarlos a tasas de red (también a 10 Gbps)**.

Sin embargo, en este experimento la velocidad de escritura se vio afectada cuando se realizaron operaciones de lectura en el RAID, ya que el límite superior del 50 % de velocidad de escritura en el RAID se vio excedido (para ser capaces de escribir a 10 Gbps), efecto que se constató en la sección 3.4.2.

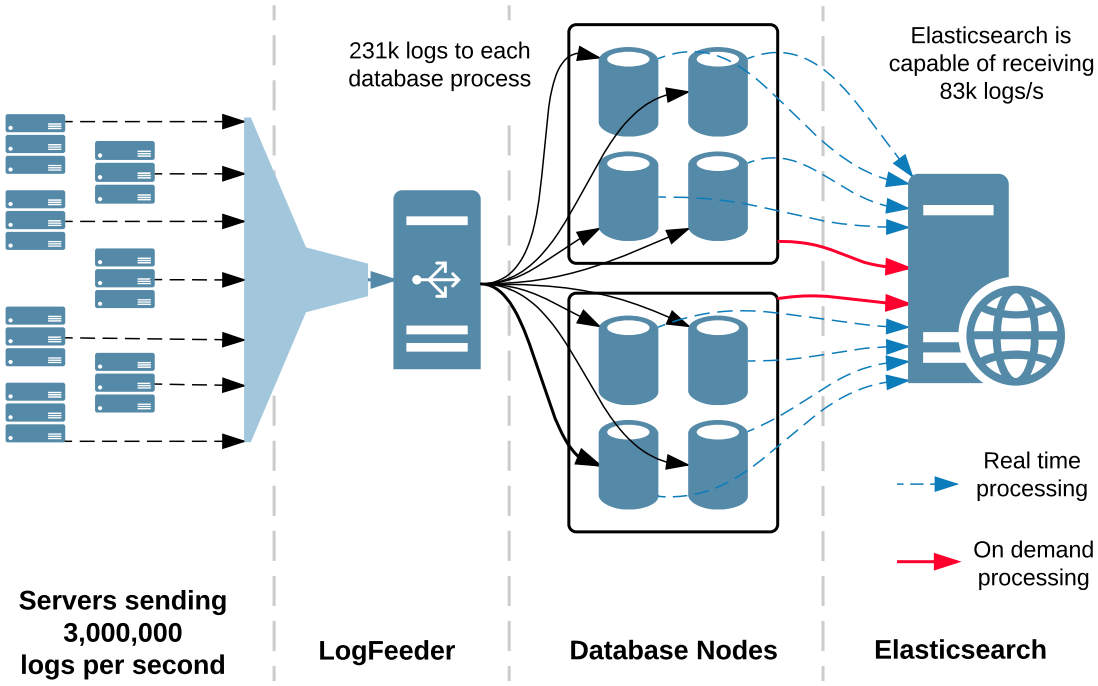



Figura 3.12: Sistema completo

## 3.5. Conclusiones

En esta investigación se presentó una solución de alto rendimiento para la recolección, almacenamiento, procesamiento y visualización de mensajes de *logs* llamada *Loginson*, el cual busca un equilibrio entre simplicidad y rendimiento, consiguiendo una velocidad notable de 3 millones de mensajes de *log* por segundo para el tamaño de mensaje más común y hasta 10 millones de *logs* por segundo para mensajes pequeños. Además, todos los componentes de la solución propuesta son independientes, lo cual hace posible el escalado horizontal para conseguir velocidades mayores. De hecho, como se verá más adelante, parte del desarrollo y conceptos presentes en este proyecto han sido empleados para el despliegue de un sistema de monitorización en la *Sociedad Estatal de Correos y Telégrafos* de España, como parte de un proyecto de *Naudit* comenzado en 2015, y que a día de hoy sigue evolucionando y aportando información esencial para la resolución de incidencias.

Todo este trabajo se publicó en la revista *The Journal of Supercomputing* bajo el título *Loginson: a transform and load system for very large scale log analysis in large IT infrastructures* [69] en el número 9 de Septiembre de 2017 (volumen 73), estando previamente disponible en la versión digital de la revista el 6 de Marzo de 2017.

Como trabajo futuro se propuso mejorar el sistema de pre-procesamiento, ya que el sistema actual es únicamente capaz de transformar los datos para su consumo en Elasticsearch. Tal sistema sería capaz de analizar mínimamente los mensajes y extraer únicamente la información relevante, detectando anomalías de forma rápida y eficiente en intervalos cortos de mensajes.

Pencroff corrió a su campo de trigo.
La isla misteriosa, Julio Verne. 1874

CAPÍTULO 4

# *FERMIN*: AUTOMATIZANDO EL ANÁLISIS DEL TRÁFICO

En esta investigación se presenta em FERMIN, una solución para el análisis del tráfico y la generación automática de informes orientada a la mejora proactiva de la monitorización en grandes infraestructuras de TI. *FERMIN* provee un sistema de detección proactiva de anomalías con el que reportar al administrador de TI un resumen relevante con los eventos más interesantes y anómalos que han ocurrido en la red. Esta información facilita al administrador de TI la resolución de problemas y el despliegue de medidas correctivas en los sistemas de monitorización y alarmas.

En lo que a rendimiento se refiere, esta propuesta es capaz de procesar grandes volúmenes de tráfico a partir de registros enriquecidos obtenidos de diversos recolectores y disectores como los estudiados en capítulos anteriores. Su versatilidad de alto nivel permite un desarrollo rápido del sistema en *hardware* convencional, el cual es esencial para enfrentarse a la creciente demanda de escenarios en los que se requiere el análisis tráfico de forma versátil y rentable.

Este estudio aborda varios retos del análisis masivo de datos mediante el desarrollo de módulos de alto rendimiento que fusionan el potencial de los lenguajes de alto rendimiento con la versatilidad de los lenguajes de alto nivel.

## 4.1. Introducción a la detección de anomalías

Los cortes en el servicio son una de las mayores preocupaciones de cualquier centro de datos o administrador de redes. En una muestra de 69 centros de datos de 43 instituciones, resultó que el coste medio por minuto de los cortes de servicio en centros de datos era de 7.908 dólares americanos (USD) [70]. Con el fin de prevenir estos incidentes, el análisis del tráfico resulta una actividad fundamental en la administración de un centro de datos, el cual ayuda a identificar de forma proactiva las potenciales fuentes de problemas, antes de que desencadenen problemas.

El carácter de alta velocidad de las actuales redes, junto con el creciente número de sistemas y aplicaciones, hace del análisis del tráfico una tarea cada vez más difícil. Ya no es cuestión de unos pocos MBytes de tráfico mediante un *sniffer* de tráfico, sino ser capaz de asimilar cientos de GBytes de tráfico y proveer un diagnóstico rápido del incidente o un análisis proactivo. Este proceso consiste principalmente en canalizar los paquetes entrantes en registros de flujo, los cuales, son posteriormente usados para generar un informe.

A medida que el ancho de banda de las redes se incrementa, el numero de flujos generados crece y el poder de procesamiento requerido para extraer la información clave a partir de estos registros y resumirlos en un informe legible, se está convirtiendo cada vez más en un reto. En el pasado, la cantidad de registros de flujos era manejable en un PC convencional y el analista podía simplemente usar scripts para procesar los registros de flujos, construir los correspondientes gráficos y tablas y finalmente producir un informe. Hoy en día, la cantidad de registros de flujos ha crecido tanto que los sistemas de generación de informes automáticos deben ejecutarse en potentes servidores.

Este ultimo paso en el proceso de canalización de la información es clave para el análisis de tráfico. Huelga decir que las grandes bases de datos distribuidas, tales como los sistemas Hadoop, pueden ser usadas con este fin, sin embargo, este enfoque supone un gran coste en *hardware* y resulta impracticable su despliegue en una red real, teniendo en cuenta que el espacio para los servidores y el gasto eléctrico están limitados. Por tanto, el enfoque seguido en este estudio es el de aprovechar la escalabilidad vertical y adoptar un sistema multi-core como plataforma de procesamiento en vez de un sistema completamente distribuido de almacenamiento y procesamiento. Esta estación de trabajo puede también ser usada como sonda de captura de tráfico y por tanto, ser usada para el proceso de captura, análisis y generación de informes.

En este trabajo se consideró una gran infraestructura de TI (centro de datos, gran red, etc.) la cual realiza una gestión continua del servicio de TI de acuerdo a lo estipulado en la

metodología ITIL. Con este fin, una evaluación del riesgo (también conocido como análisis de riesgo) es realizada con el fin de identificar activos, amenazas, vulnerabilidades y contramedidas para cada servicio de TI. Como parte de dicha evaluación, se realiza un análisis del tráfico de manera diaria o semanal, mediante un sistema automatizado de informes de análisis del tráfico. Posteriormente, se implementan las medidas correctivas necesarias y se actualizan los sistemas de alarmas y de monitorización en consecuencia.

Debe destacarse que el primero se trata de **sistemas de monitorización reactiva**, los cuales, en caso de un incidente, proveen **información microscópica** sobre ciertos indicadores clave del rendimiento (KPI en sus siglas en inglés, *Key Performance Indicators*). Típicamente, estos están basados en consultas SNMP continuas y sirven para destacar eventos como una alta utilización de CPU o del enlace.

En la práctica, establecer umbrales resulta todo un reto, ya que los sistemas de redes los puntos de operación cambian con bastante frecuencia. Además, gran parte de los incidentes ocurren como consecuencia de un problema latente, el cual crece sin ser detectado hasta que la degradación del servicio ocurre. Precisamente, esta clase de problemas subyacentes son el objetivo del análisis de tráfico automatizado, el cual es mayormente proactivo. De hecho, el análisis de riesgos es una **tarea proactiva de largo plazo**, el cual está basado en un **análisis macroscópico** del tráfico y los mensajes de *logs*. Estos dos enfoques, reactivo y proactivo, están pensados para ser complementarios y servirse el uno al otro. Este capítulo, se centrará en el análisis proactivo del tráfico.

## 4.2. Estado del arte

El estado del arte se enfoca en sofisticadas técnicas para la detección o correlación de problemas. Entre otros, Luo et alii en [71], proponen técnicas para la detección de correlaciones entre series temporales (uso de CPU) y una serie de eventos (comienzo de un programa que consume mucha CPU), centrándose en la existencia de correlaciones temporales y espaciales. Otras propuestas, como las presentados por Bahl et al. [72] o Ciraci et al. [73], despliegan agentes para el análisis de datos en las máquinas clientes, lo cual requiere un entrenamiento previo de los agentes.

Mientras que este estado del arte provee técnicas de gran valor para el análisis del tráfico, es necesario remarcar que el caso de uso expuesto considera una cantidad mucho mayor de datos y que su enfoque es mucho más macroscópico. En la experiencia adquirida durante los proyectos, un gran centro de datos produce varios TBytes de tráfico a analizar por día [74], los cuales deben ser tenidos en cuenta de forma rentable, es decir,

con el menor número de recursos posibles. Incluso cuando usan registros enriquecidos en vez de paquetes en crudo, el volumen de datos alcanza las decenas de GBytes.

Este requisito impone un compromiso entre la relevancia de los hallazgos, el tiempo de procesamiento y los recursos utilizados. Si los recursos para realizar el procesamiento son abundantes, lo cual no es lo usual, entonces la profundidad del análisis puede ser mayor. Sin embargo, tal nivel de detalle puede también ser contraproducente, ya que el personal de TI siempre tiene una apretada agenda la cual únicamente le permite centrarse en los eventos más relevantes. Como indican Godall et al. [75], *"los analistas suelen perder la vista de la imagen global mientras examinan los pequeños detalles."* (*"analysts often lose sight of the 'big picture' while examining these low-level details"*).

En este capítulo, se presenta *FERMIN* (*Factual Executive Report of a Monitored IP Network*)[1], un sistema para la generación automática de informes de red. *FERMIN* provee al administrador de red con un informe con los eventos más relevantes todo ello en el menor tiempo posible y con unos requisitos de procesamiento limitados. A continuación se desgranan los detalles y principios de diseño de *FERMIN*, así como las lecciones aprendidas durante el proceso, que puedan ayudar a investigadores y profesionales del área a construir sistemas eficientes para el reporte automático de informes de red.

## 4.3. Auditando grandes redes

El caso de uso propuesto comprende un gran centro de datos, en el cual se dispone de una sonda de análisis capaz de recolectar tráfico a altas velocidades. La figura 4.1 muestra las diferentes etapas para la generación de informes automáticos. En la etapa de recolección de datos, una sonda de tráfico recibe una copia del tráfico de ciertas VLANs, subredes, o puertos de interés a través de un SPAN o un puerto espejo que redirige el tráfico a la sonda. Entonces, múltiples disectores se aplican en el tráfico capturado con el fin de generar registros enriquecidos, los cuales corresponden a los diferentes protocolos presentes en el tráfico. Por ejemplo, los registros HTTP que contienen el cliente, servidor, URL, tiempo de respuesta, código de respuesta, etcétera. Finalmente, el informe es generado usando estos registros enriquecidos como entrada de datos.

Las etapas anteriores pueden ser realizadas en un único nodo o múltiples, ya sea en serie o paralelo. Es recomendable usar solamente uno siempre que sea posible. Si se usan múltiples nodos en serie, repercutirá en la cantidad de datos a mover entre uno y otro. Por ejemplo, si un nodo es empleado únicamente para la recolección de datos y la

[1] `http://www.naudit.es/fermin`

disección, mientras que un segundo nodo es utilizado para el análisis y la generación de informes, todos los registros enriquecidos deberán ser movidos desde el primer nodo al segundo. En cuanto al despliegue en paralelo, es decir, mediante un balance de carga, es necesario advertir del reto no trivial que supone la distribución uniforme de los datos. Por ejemplo, Shi et al. concluyen que *"bajo ciertas distribuciones de los tamaños de flujo, que siguen un tipo Zipf, no es posible distribuir de forma balanceada mediante esquemas basados en hashing"* [76] (*"under certain Zipf-like flow-size distributions, hashing alone is not able to balance workload"*), en relación a los flujos IP.

Es más, el coste también se incrementa cuantos más nodos se usan. Por tanto, en esta investigación se trata de exprimir la máxima eficiencia en las etapas de recolección de datos, disección, y análisis, con el fin de realizar todas estas etapas en el mismo nodo.

Estas etapas no tienen por qué ser realizadas necesariamente de manera concurrente. Una traza de tráfico puede ser recolectada durante el día, con su consiguiente disección realizada durante la noche. Como se mencionó anteriormente, este sistema de informes automáticos no está orientado a la operación en tiempo real y las alarmas, sino al análisis proactivo a largo plazo.

Precisamente, el análisis proactivo a largo plazo tiene un rol importante en la planificación de los recursos. Durante el día, en las horas pico los recursos escasean, y por lo tanto los sistemas reactivos en tiempo real deben estar en continua búsqueda de problemas, cortes o incidencias. Estos sistemas en tiempo real no pueden optar por tareas futiles y deben estar bien orquestados y optimizados para usar tan pocos recursos como posibles, y al mismo tiempo, ser efectivos. El análisis proactivo realizado durante la noche o los fines de semana ayuda a ajustar y dirigir los sistemas de tiempo real, ahorrando tiempo y recursos durante el día, incrementando su efectividad.

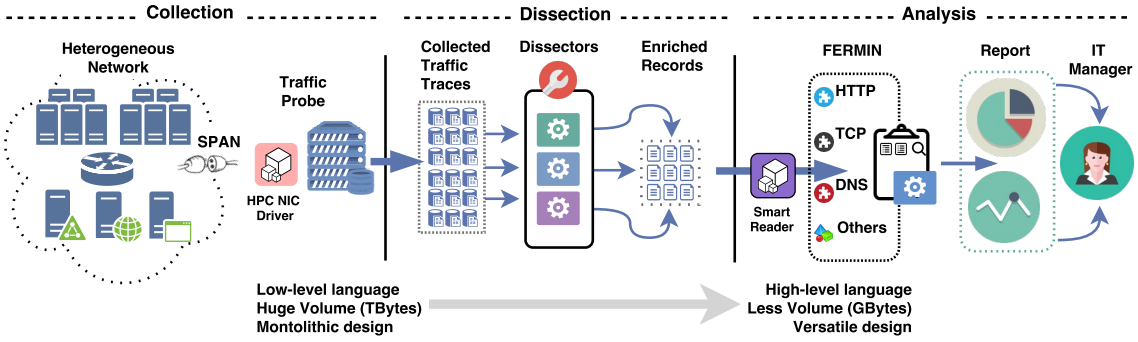



Figura 4.1: Despliegue típico para la recolección y análisis de tráfico, mostrando las diferentes etapas involucradas en el proceso.

En lo que sigue, se explicará en profundidad cada una de las etapas anteriormente mencionadas. En concreto, se cubrirán los siguientes asuntos.

1. Captura del tráfico a velocidades de multi-gigabit por segundo y la generación de registros enriquecidos y estadísticas agregadas haciendo uso de disectores de alta velocidad.

2. La selección de indicadores clave del rendimiento.

3. Análisis de la información a través de diferentes módulos de software.

4. La generación automática del informe con tablas y diferentes visualizaciones.

### 4.3.1. Captura de tráfico y generación de registros enriquecidos

Primero, el tráfico en las redes corporativas debe ser capturado ya sea a través de *hardware ad-hoc* o controladores optimizados capaces de recibir tráfico de red a velocidades de 10 Gbps tales como las que soporta el controlador HPCAP [31] para tarjetas de red Intel.

Se observa que el envío de registros obtenidos a partir de las conexiones de tráfico es una tarea extremadamente difícil, si no imposible, a realizar a altas velocidades, especialmente si se pretende inspeccionar la capa de aplicación. En consecuencia, se considera como punto de partida para la generación de informes no es tanto el cuerpo de registros enriquecidos producido en tiempo real, sino la traza de tráfico real en sí misma, la cual es subsecuentemente procesada para obtener registros enriquecidos.

Entonces, el primer paso en la generación de informes es la disección de la traza del tráfico. Los disectores de tráfico proveen registros para cada servicio (HTTP, DNS, SQL, SMB, etc.), por protocolo (TCP, UDP, ICMP), etc. Es más, también proveen series temporales que muestran la evolución temporal de un estadístico en concreto, por ejemplo, el tráfico en bits por segundo, para una VLAN o subred en particular. En el estado del arte hay disponibles numerosos disectores de alta velocidad [32, 77] (como los tratados en capítulos anteriores) por lo que no nos centraremos en ellos durante este capítulo.

### 4.3.2. Clasificación de los KPIs

La elección de KPIs es un debate abierto ya que las preocupaciones de los administradores de TI pueden ser encaradas de diversas formas. En general, los KPIs pueden ser clasificados entre con y sin estado.

#### 4.3.2.1. KPIs sin estado

Estos KPIs usualmente consisten en contadores absolutos que no son conscientes del contexto, es decir, realizar estas estimaciones no requiere guardar un estado previo que repercuta en el resultado de una entrada futura. Típicamente, estos informan de valores globales tales como el número de paquetes o bytes retransmitidos. Proveer estas métricas en línea es posible incluso a altas tasas debido a que no requiere de grandes recursos.

#### 4.3.2.2. KPIs con estado

Por otro lado, otras métricas más elaboradas requieren consciencia previa de los estados previos, tales como los relacionados con las conversaciones, conexiones o sesiones. Estos KPIs requieren mayores cantidades de recursos y consumen más tiempo de procesamiento, poniendo en peligro la entrega de estas métricas en tiempo real. En el peor caso, estos pueden requerir múltiples pasadas sobre los datos analizados.

Por ejemplo, las sondas de tráfico producen registros enriquecidos en tiempo real que proveen el número de anuncios de ventana cero a partir de un cliente o servidor dado en un flujo TCP. Sin embargo, obsérvese que no sólo los anuncios de ventana cero importante, sino el tiempo máximo en que el cliente o el servidor no estaba abriendo esta ventana. Si un cliente o servidor anuncia una ventana cero solamente una vez durante una conexión TCP, entonces no es muy relevante. Por el contrario, si este anuncio se mantiene durante 10 segundos, existe la posibilidad de que el servidor esté altamente saturado. Sin embargo, el tiempo de bloqueo (v. gr., no abrir la ventana) deberá ser obtenido fuera de línea con el fin de reducir los requisitos de procesamiento de la sonda durante la captura y almacenamiento del tráfico a altas velocidades.

## 4.4. Principios de diseño

A partir de lo explicado en la sección anterior, se pueden resumir algunos requisitos esenciales que el sistema debe ser capaz de satisfacer. Adicionalmente, en esta sección se describirán las decisiones de diseño tomadas durante el desarrollo de este sistema, así como sus características. Dicho esto, a continuación se describen todas ellas.

### 4.4.1. Tratando grandes volúmenes de datos

La velocidad de procesamiento es un requisito crucial para la generación de informes automáticos. En lo que respecta a la implementación, los aspectos más importantes en la generación de informes automáticos son el análisis en una única pasada y disposición de los datos en memoria, los cuales, imponen claras restricciones en los KPIs que vayan a ser incluidos en el informe.

El **análisis en una única pasada** acarrea que la lectura de la traza de tráfico original sea leída de disco al menos una vez, diseccionando toda su información en registros, y, produciendo entonces los gráficos y tablas necesarios para el informe. Es necesario advertir que leer TBytes de datos desde disco conlleva mucho tiempo.

En consecuencia, cuantas menos pasadas se realicen sobre la traza de datos original, mejor rendimiento final conseguiremos en términos de velocidad. Por tanto, la mayoría de los KPIs deberán ser obtenidos a partir de los registros enriquecidos. Por ejemplo, obtener una serie temporal detallada de una máquina en concreto, la cual haya sido identificada como interesante a partir de los registros enriquecidos, implicaría la lectura de la traza de tráfico por segunda vez.

En cuanto a la **disposición de los datos en memoria**, es una tarea que comprende la extracción de los campos de interés a partir de los registros enriquecidos y mantenerlos en memoria. Como se ha indicado anteriormente, en un gran centro de datos, un día de tráfico ocupa entre 2 y 3 TBytes de datos y los registros enriquecidos derivados pueden ocupar varias decenas de GBytes. Con el fin de obtener los gráficos y tablas del informe, estos registros enriquecidos serán altamente consultados durante la creación del informe. Por tanto, estos deberán estar cacheados en memoria tanto como sea posible, evitando su re-lectura de disco con el fin de evitar penalizar el rendimiento.

Teniendo todo esto en cuenta, es necesario realizar un plan de memoria en la máquina de forma cuidadosa. Para ello, resulta fundamental conocer el volumen aproximado de registros enriquecidos por TB de traza de tráfico.

La figura 4.2 muestra el volumen de registros enriquecidos por TB de traza de tráfico para dos centros de datos diferentes, para flujos MAC/IP/TCP/UDP/HTTP con una longitud de registro medio de $29 \pm 44$ campos (métricas) y $195 \pm 121$ bytes por registro. La longitud en bytes de los registros enriquecidos abarca desde menos de 10 para el caso de MAC hasta más de 1000 para el caso de TCP. El primero consiste generalmente en unos pocos contadores del número de paquetes o bytes transmitidos entre dos extremos, mientras que el segundo de ellos contiene cientos de métricas para ambos sentidos, desde el número de retransmisiones, hasta el máximo tiempo entre dos segmentos TCP.

La figura muestra que para un gran centro de datos varias decenas de millones de registros pueden ser producidos por día. Por tanto, será necesario encontrar un compromiso entre la versatilidad dada por la programabilidad de alto-nivel y la eficiencia del proceso a realizar.

### 4.4.2. Análisis de los KPIs a través de estadísticas de alto nivel

Como se ha indicado anteriormente, la selección de KPIs está condicionada por los susodichos factores limitantes del análisis en una única pasada y la disposición de los datos en memoria. Huelga decir que los indicadores básicos de volumetría (principales clientes, servidores, conversaciones, puertos, etc.) pueden ser obtenidos fácilmente a partir de los registros enriquecidos usando tablas *hash* almacenadas en memoria. Otros KPIs más elaborados pueden requerir varias pasadas de los registros enriquecidos para la obtención del resultado, lo cual podría penalizar el rendimiento.

No se abundará en una descripción de todos y cada uno de los KPIs por razones de brevedad. En cambio, se describirán algunos de los KPIs utilizados los cuales acatan los principios de diseño anteriormente explicados, y que además proveen de valiosa información al administrador de TI.

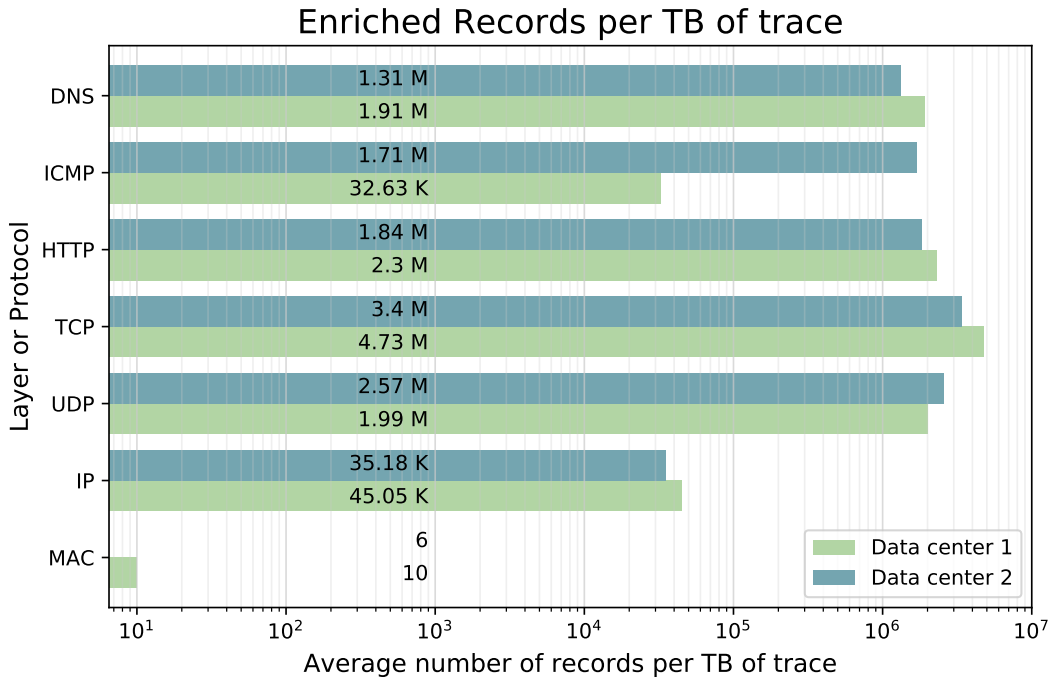



Figura 4.2: Registros enriquecidos por TB de traza en distintas redes empresariales. Las etiquetas indican el valor de cada barra.

#### 4.4.2.1. Análisis de ráfagas e identificación de su causa raíz

Con el fin de identificar ráfagas (ver figura. 4.3) o mesetas, se hace uso de una serie temporal de los bits por segundo de la traza completa, la cual es obtenida a partir de la primera pasada de la traza. Para ello se adopta un heurístico simple para la detección de ráfagas basado en un umbral de la velocidad alcanzada y la duración de la ráfaga. Después, se provee una posible causa, la cual es de interés para identificar lo ocurrido.

Con este fin, el algoritmo consiste en probar un conjunto de series temporales de distintas métricas (tiempo de respuesta HTTP, retransmisiones TCP, etc.) contra cada una de las ráfagas encontradas a partir de la serie temporal de bits por segundo de la traza. Para ello se buscan correlaciones en la variación de las métricas. La métrica con la mayor variación en el intervalo en el que ocurre la ráfaga es la elegida como candidata para la causa raíz, y los extremos responsables de haber producido esta segunda serie temporal son destacados en el informe.

Por ejemplo, la figura 4.5 muestra un ejemplo de ráfaga de tráfico de un informe real. Las ráfagas están marcadas mediante un área amarilla, y en este caso la causa candidata de la ráfaga de tráfico es un incremento del tráfico por conexión.

#### 4.4.2.2. Reconstrucción de series temporales a partir de los registros enriquecidos

En este punto, es clave entender que no es posible obtener una serie temporal precisa a partir de los registros enriquecidos [78]. Por ejemplo, si quisiéramos obtener el tráfico en bits por segundo entrantes y salientes para una IP y puerto específicos, los registros


Figura 4.3

Ejemplos de detección de ráfagas: Ráfagas detectadas (verde), ráfagas extendidas (amarillo puntuado) y ráfagas rechazadas por cortas y/o su velocidad media. (rojo rayado). En naranja se muestran los umbrales de detección.

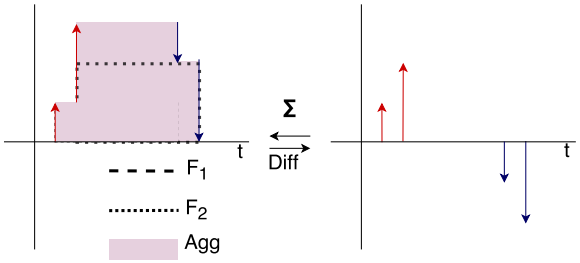



Figura 4.4

Reconstrucción de una serie temporal. La figura representa un ejemplo con dos flujos, en la cual la reconstrucción es realizada mediante la suma cumulativa de los impulsos.

enriquecidos sólo nos proveerían la media de tráfico por flujo, para aquellos flujos que tienen la misma IP y puerto como uno de sus extremos.

Por lo tanto, tan sólo es posible recrear la serie temporal buscada mediante la adición de esas medias a modo de funciones escalonadas en los intervalos donde el flujo está presente (ver figura 4.4). Este método asume que el ancho de banda consumido durante la duración del flujo es uniforme, lo cual, generalmente, no es cierto. El impulso positivo inicial y el impulso negativo final podrían ser considerados como la derivada de esta distribución uniforme, por tanto, aplicando la operación de suma cumulativa se estaría integrando esta serie de impulsos, resultando en una serie temporal aproximada para esta métrica específica. Sin embargo, esto no es, ni de lejos, la serie temporal del tráfico entrante y saliente de ese conjunto de IPs y puertos.

La serie resultante de la composición de esas funciones escalonadas es mucho más suave y menos variable que la serie temporal real. Por tanto, no se pueden mostrar los picos de tráfico de forma precisa a partir de ella, lo cual es clave para algunos servicios. Por ejemplo, resulta esencial para la detección de picos en el tráfico Citrix [79], ya que estos picos producen retardos instantáneos de encolamiento, los cuales son apreciables por los usuarios incluso en pequeñas escalas de tiempo.

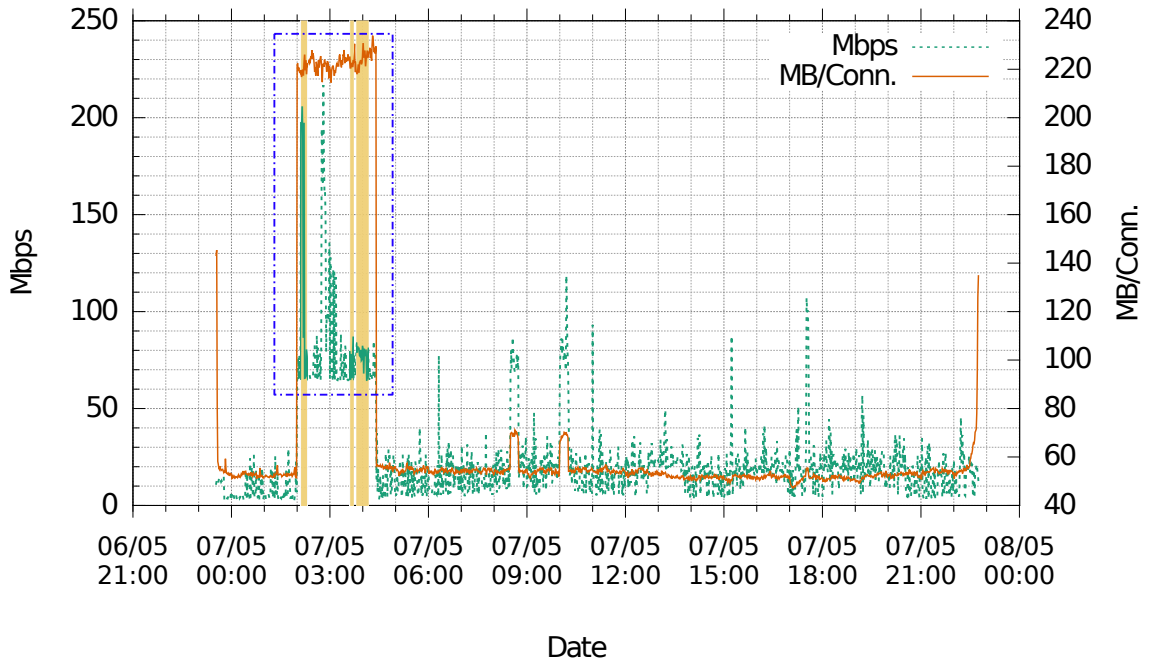



Figura 4.5: En el intervalo marcado con un rectángulo azul, una ráfaga de tráfico (verde puntuado) es causado por un súbito incremento del tráfico por conexión (naranja).

En este ejemplo, se observa que la serie temporal a obtener no puede ser programada *a priori*. Primero, se realiza un análisis a nivel de flujo o conexión el cual arroja las máquinas con problemas potenciales (v. gr., un gran ratio de retransmisiones por flujo). Después, se puede obtener una serie temporal detallada para esa máquina en particular con el fin de encontrar las posibles anomalías en el tráfico, tales como micro-cortes en la escala de segundos los cuales no pueden ser derivados directamente a partir de los registros enriquecidos.

#### 4.4.2.3. Resúmenes RAG

Un resumen RAG (Rojo, ambar, verde, o en inglés, *Red-Amber-Green*) ofrece una evaluación a alto nivel de un sistema, servicio o protocolo específico. Con este fin, se realizan resúmenes de determinadas métricas para distintas capas y protocolos como DNS, TCP, HTTP, etc. Por ejemplo, para un RAG del estado del protocolo TCP las siguientes métricas de TCP son tenidas en cuenta: anuncios de ventana cero, retransmisiones, ACKs duplicados, intentos de conexión, cortes, RTT, tiempo de establecimiento de conexión y el número de conexiones junto con su contribución relativa en bytes al tráfico TCP.

Varios umbrales y heurísticos son empleados para caracterizar la salud de un servidor en diferentes grupos, como por ejemplo, si el intervalo de tiempo entre el SYN-ACK y el ACK durante el establecimiento de la conexión TCP es mayor que 1 segundo durante 5 minutos consecutivos, o si ocurre una gran desviación sobre la media (v. gr., un incremento de 10 veces en el RTT), estos se considerarían comportamientos anormales para el caso de la métrica del RTT.

Tras este resumen, se obtiene información en detalle para los servidores críticos de cada análisis RAG, reconstruyendo las series temporales para las métricas afectadas o realizando otros análisis de interés.

Otros muchos KPIs ofrecen información valiosa para el administrador de IT. Por ejemplo, el análisis de topología tanto para el nivel de enlace (MAC) como para la capa de red (IP), o métricas locales tales como el ratio de retransmisiones por flujo (o conexión) de los servidores, o el número de clientes únicos de un servidor, los cuales reciban respuestas ICMP de un tipo en concreto. Gracias al sistema modular estos pueden ser extendidos o modificados a medida.

### 4.4.3. Versatilidad para un desarrollo rápido

Las necesidades de las etapas de recolección, disección y análisis son claramente diferentes desde el punto de vista de la programación. Según vamos escalando desde el cable hasta los gráficos del informe (ver figura 5.2), tendemos a agregar al información, desde los paquetes individuales a los registros enriquecidos, y finalmente los gráficos en el informe automático. Durante el proceso de recolección, *software ad-hoc* a nivel de núcleo ofrece el máximo rendimiento para evitar la pérdida de paquetes durante el proceso de captura. Después, mediante disectores de alta velocidad se extrae la información, también de forma rápida, pero permitiéndose ignorar ciertas restricciones que previamente eran vitales (v. gr., los controladores deben priorizar no perder paquetes).

Estas dos primeras etapas usualmente priorizan requisitos no funcionales tales como el rendimiento, eficiencia, estabilidad o disponibilidad, y tienden a evolucionar lentamente con pocos cambios durante su ciclo de vida debido a su diseño ad-hoc y monolítico. Por ejemplo, los disectores de protocolos de tráfico no son modificados muy a menudo ya que los protocolos (v. gr. TCP) no son tampoco modificados frecuentemente.

Por otro lado el proceso de análisis requiere un nivel de abstracción mayor y funciones complejas con mayor tendencia a cambiar según las necesidades del analista o el administrador de TI. Estos son generalmente requisitos funcionales tales como analizar nuevas métricas o dibujar gráficas adicionales. Por tanto, la solución *software* para estas etapas generalmente requiere requisitos no funcionales tales como modularidad, extensibilidad o versatilidad.

#### 4.4.3.1. Análisis de los registros enriquecidos

Por tanto, con el fin de enfrentar los diversos requisitos de análisis mencionados anteriormente, el sistema *FERMIN* fue desarrollado en *Python* debido a su gran versatilidad y portabilidad, haciendo uso de librerías estadísticas de alto nivel tales como *Pandas*[2] and *Numpy*[3], la cual facilita un rápido desarrollo del sistema a la vez que mantiene un buen rendimiento. La capacidad de desplegar un sistema de informes automatizados sobre el tráfico como *FERMIN* en *hardware* convencional lo hace adecuado para coexistir con otras tareas esenciales, tales como los sistemas de alarmas y monitorización, complementándose y sirviéndose los unos a los otros.

Lenguajes como C# o Java fueron considerados, no obstante, sería más complejos de desplegar junto con sus correspondientes dependencias en sistemas Linux. Lenguajes

[2] `http://pandas.pydata.org/`
[3] `http://www.numpy.org/`

compilados tales como C o C++ hubieran ofrecido mayor rendimiento pero a un coste de desarrollo muy alto. Sin embargo, se puede encontrar un compromiso que evite un excesivo tiempo de desarrollo y que obtenga a su vez las ventajas de rendimiento y uso de recursos del lenguaje C haciendo uso de la interfaz de C-Python en aquellas secciones críticas. Se ha demostrado extremadamente útil, ya que Python tiene un rendimiento muy pobre en términos de uso de memoria y velocidad a la hora de leer ficheros. Alternativas ya mencionadas como Pandas o Numpy no podían soportar todos los tipos de datos considerados. Por lo tanto, se desarrolló un módulo de C, (*cparser*[4]) que permite la lectura rápida de registros con un bajo uso de memoria en comparación con el rendimiento ofrecido por Python. Este módulo ha sido liberado para la comunidad estando disponible su código en un repositorio de código público.

La figura 4.6 muestra una comparativa entre el rendimiento de *cparser* y las librerías estándar de Python para entrada/salida. La figura muestra los tiempos de lectura de un millón de registros, ejecutados diez veces por experimento. Como se puede observar, la reducción en tiempo de lectura es notable. Esta mejora de rendimiento significa que el tiempo empleado durante el análisis de los registros (la lectura inicial) es aproximadamente un 1% del tiempo total de ejecución (8 minutos). Sin *cparser*, el informe del conjunto más grande de datos utilizado en este estudio que duró 9 horas y 30 minutos, hubiera necesitado un día completo de ejecución. Todo esto sin tener en cuenta la reducción adicional de rendimiento durante las etapas de filtrado ni los problemas causados por un mayor uso de memoria.

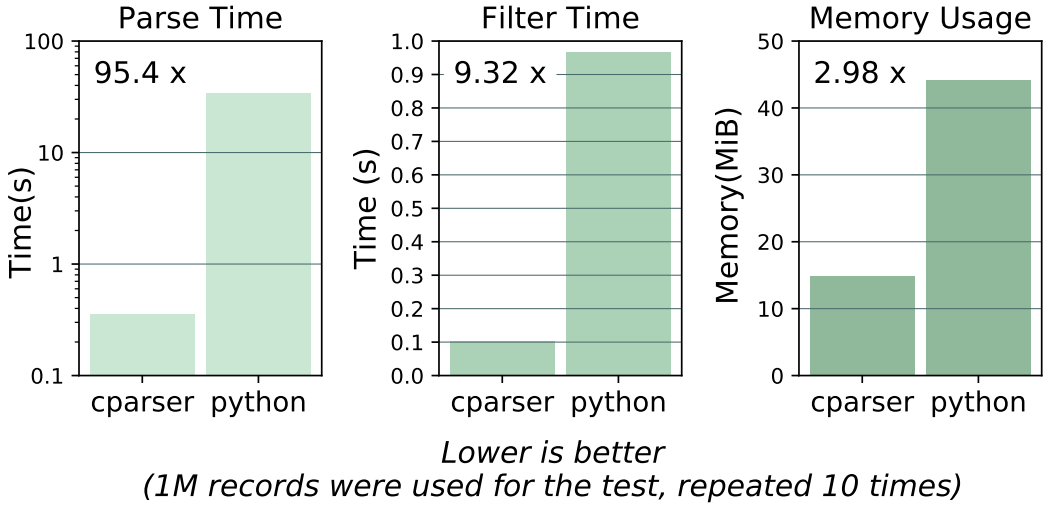



Figura 4.6: Comparativa de rendimiento entre Cparser y Python.

[4]https://hpcn-uam.github.io/ccore/

En un nivel superior, varios módulos han sido desarrollados para el análisis de cada tipo de tráfico y capa, permitiendo añadir nuevos protocolos al sistema en el futuro. Estos módulos procesan información obtenida a partir de varios disectores de tráfico disponibles en la caja de herramientas de *Naudit*[5] y pueden centralizar información común en el módulo *Network Information* con el fin de reunir información común entre capas y protocolos. Este módulo es también el encargado de establecer filtros comunes en todas las capas usando un pequeño subconjunto de reglas similar a los filtros *Berkeley Packet Filters* [22], lo que permite filtrar los registros basándose en rangos de IPs, subredes o puertos.

También se desarrolló un módulo de lectura *smartReader* que ayuda en la creación de una cadena de operaciones tales como filtrar, agregar, top (generar un ranking de claves en base a un valor), etc., encadenados juntos de forma similar a la realización de una consulta sobre los datos. Este módulo ayuda también a cachear los datos y aplazando las operaciones de lectura hasta que son necesarias. Por debajo, un módulo de lectura desarrollado en lenguaje *C* ofrece una conversión rápida de los tipos y una creación eficiente de los objetos a ser leídos.

#### 4.4.3.2. Visualizaciones y generación del informe

Para generar visualizaciones útiles y prácticas, basta con seguir unas reglas simples [80] acerca de cómo representar los datos de manera apropiada. Debido a la gran variedad de gráficas y de librerías de visualización disponibles, no resulta trivial elegir cómo mostrar la información, especialmente cuando se busca automatizar todo el proceso. Se estudiaron y probaron extensamente distintas soluciones para la visualización de datos. Finalmente se optó por algunas de las librerías más populares para la visualización de datos, tales como *gnuplot*[6], *tikz*[7] o *matplotlib*[8], debido principalmente a su buena integración, versatilidad y posibilidades de personalización. Para el caso de la visualización de datos, se implementó una variedad de gráficas personalizadas tales como gráficos para la función de supervivencia, *boxplots*, series temporales, gráficos de tartas y grafos de topologías. Todas estas visualizaciones se adaptan a los datos para, por ejemplo, permiten remuestrear las series temporales de acuerdo al intervalo de tiempo que muestran.

[5] `http://www.naudit.es/en/`
[6] `http://www.gnuplot.info/`
[7] `http://pgf.sourceforge.net/`
[8] `https://matplotlib.org/`

Finalmente, el informe es producido agrupando todo este material, mediante un lenguaje de maquetación de textos como es LaTeX, el cual es código abierto y permite realizar cambios *a posteriori*, produciendo un documento con formato a partir de ficheros de texto. Gracias al diseño modular de *FERMIN*, no se descartan otros formatos de salida tales como texto plano por consola, el cual podría conectar los resultados de los informes a un sistema de herramientas de administración.

#### 4.4.3.3. El futuro de los lenguajes de alto nivel para los sistemas de informes automáticos

Se espera que el progreso de varios proyectos tales como PyPy[9], Numba[10] o Grumpy[11], impulse el desarrollo de sistemas versátiles con gran rendimiento. Estos proyectos tienen como objetivo diluir las barreras entre los lenguajes de alto y bajo nivel, uniendo la fuerte abstracción de los primeros con el alto rendimiento de los segundos. Es necesario indicar que estos proyectos todavía están en desarrollo y que deben todavía ampliar su compatibilidad con muchas librerías, pero existe una clara tendencia hacia herramientas de alto rendimiento de análisis con lenguajes de alto nivel **??**. Muchos de estos proyectos trabajan con lenguajes de alto nivel y compiladores en tiempo de ejecución (también conocida por sus siglas en inglés, JIT, *just-in-time*), lo cual parece prometedor de cara a enfrentar las futuras necesidades de la generación de informes automáticos y análisis de datos.

## 4.5. Metodologías para el análisis automatizado del tráfico

A partir de las premisas explicadas en las secciones anteriores, se desarrolló *FERMIN*, el cual ha sido desplegado en varios proyectos con distintos clientes de *Naudit*[12]. Para evaluar *FERMIN* se recolectaron dos grupos de trazas de tráfico mediante sondas de tráfico situadas en redes de distintas corporaciones multinacionales.

Estos dos grupos de trazas tienen características muy diferentes en cuanto al tráfico. Los 6.9 TBytes (10.036 millones de paquetes) de trazas de tráfico del conjunto de datos **dataset 1** fueron capturados en dos interfaces diferentes en el borde de una red

[9] `https://pypy.org/`
[10] `http://numba.pydata.org/`
[11] `http://grump.io/`
[12] `http://www.naudit.es/fermin-es/`

empresarial, y constituyen el caso de estrés, es decir, el caso límite para el sistema, con gran cantidad de tráfico. El conjunto de datos **dataset 2** fue capturado en el núcleo de la capa de distribución de otra red empresarial, resultando en 2.1 TBytes (3.693 millones de paquetes) de trazas de tráfico, lo cual representa la cantidad media de tráfico durante un día normal.

### 4.5.1. Preparación de los datasets

Tras la recolección de las trazas de red, estas fueron resumidas en registros enriquecidos[13] mediante disectores especializados. Estos registros contienen diferentes estadísticas sobre cada flujo de la capa diseccionada y la cantidad de información es también muy variable. A pesar de que todos ellos contienen métricas en común, tales como las direcciones IP de los extremos, algunos tipos de registros tales como los referentes a conversaciones MAC o IP proveen, debido a su naturaleza, información de menor granularidad comparado con aquellos obtenidos a partir de los flujos TCP. Los protocolos de aplicación suelen contener campos basados en texto tales como la URI HTTP o el *host* solicitado durante una consulta DNS, con un impacto tanto en la longitud de los registros como en el consumo de memoria durante el análisis.

Los registros enriquecidos resultantes se ven representados por la figura 4.7, la cual muestra la cantidad de registros de cada capa o protocolo. Como era de esperar, el **dataset 1** deriva mayor número de registros que el **dataset 1** debido a su tamaño, pero la figura 4.2 también muestra como el número de registros por TByte es también muy variable. Con estos datos, se procede a la descripción de los experimentos.

### 4.5.2. Definición de los experimentos

Resulta obvio que las etapas correspondientes a las capas y protocolos que producen mayor cantidad de registros, debido a su mayor número de métricas y longitud, requieran mayor tiempo de procesamiento y memoria. Por lo tanto, el primer experimento considerado es la ejecución secuencial de cada etapa de análisis para cada conjunto de datos con el fin de encontrar qué etapas requieren más recursos tales como memoria o tiempo de procesamiento. Se ha evaluado el rendimiento de cada etapa del informe en función del número de registros haciendo uso de dos conjuntos de datos.

Estos resultados servirán para extraer conclusiones acerca de cómo algunas etapas pueden ser paralelizadas con el fin de optimizar tanto el tiempo de ejecución como los

[13] Ejemplos disponibles a petición del lector.

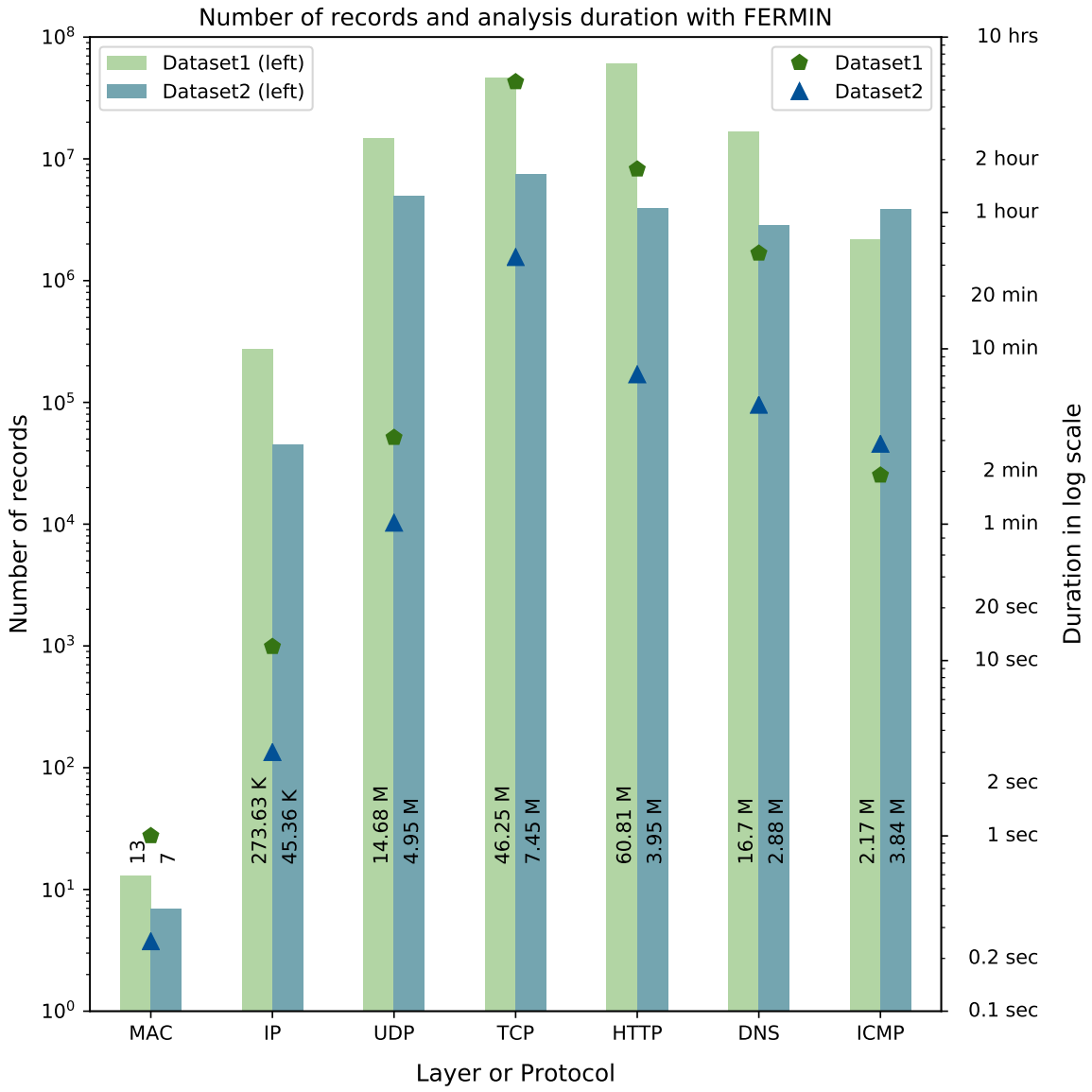



Figura 4.7: Cantidad de registros enriquecidos y sus correspondientes tiempos de procesamiento para cada capa/protocolo en diferentes conjuntos de datos.

requisitos de memoria en diferentes escenarios. Por lo tanto, el segundo experimento requiere la evaluación de diferentes planificaciones para la optimización del uso de memoria y el tiempo de ejecución, buscando un compromiso entre ambas.

Se estableció un objetivo realista, tal que sea posible procesar los registros enriquecidos correspondientes a 1 TBytes de trazas en menos de un cuarto de día, ya que, el volumen de tráfico de grandes centros de datos puede llegar a alcanzar varios TBytes de datos al día. Reducir el tiempo de ejecución con el fin de aprovechar los momentos ociosos de la sonda, tales como la madrugada o la medianoche, resulta crucial para procesar la información del día dentro de un lapso razonable de tiempo y ser capaz de desplegar las medidas correctivas, en caso de necesitarlas.

## 4.6. Evaluación del sistema *FERMIN*

Esta sección introduce la evaluación del sistema propuesto *FERMIN* para la generación de informes automáticos el cual ha sido evaluado en una máquina usando un Intel ®Xeon E5-2640 v3 @ 2.6Ghz con 128GB de RAM y dos sistemas de almacenamiento RAID 0 con 16 TBytes de espacio cada uno, los cuales proveen velocidades de escritura y lectura de hasta 10 Gbit/s, ideales para la captura de tráfico a velocidades de varios gigabits por segundo. Dicha instalación es similar a las que pueden encontrarse en las sondas de tráfico en grandes centros de datos. Solamente se ha hecho uso de un sistema RAID para almacenar los registros enriquecidos para el análisis. Es necesario destacar que el sistema propuesto, como otros tantos sistemas de análisis de redes, se ve favorecido por el escalado vertical. Por tanto, un procesador más rápido, como los disponibles a 4Ghz, podría ofrecer mejores resultados.

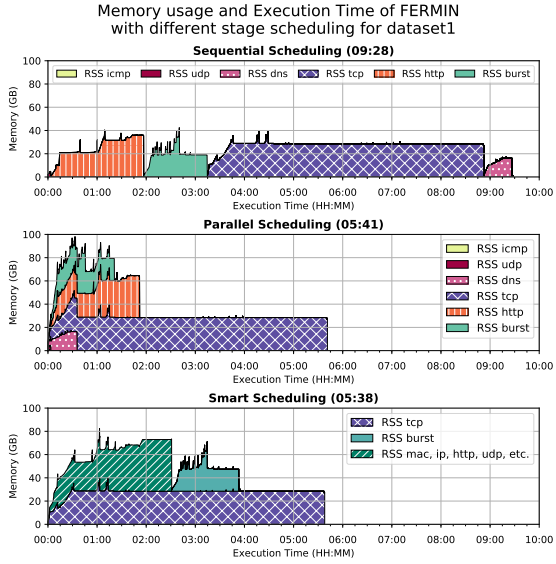
### 4.6.1. Identificación del hándicap

La ejecución secuencial del sistema confirmó (ver figura 4.7) que algunas secciones, tales como TCP, son propensas a tardar más debido a la complejidad de las métricas y su análisis. Como muestra la figura 4.8, las secciones también tienen mayor uso de memoria. Todo esto puede llevar a una clase de inanición de etapas (*stage starvation*) en las cuales las etapas más cortas esperan durante mucho tiempo en vez de ser procesadas en paralelo junto con las tareas que mayor recursos consumen o previamente a ellas. Estos resultados ayudaron en la identificación de ciertas secciones del informe que podrían ser entregadas antes si fuesen ejecutadas mediante una mejor planificación.

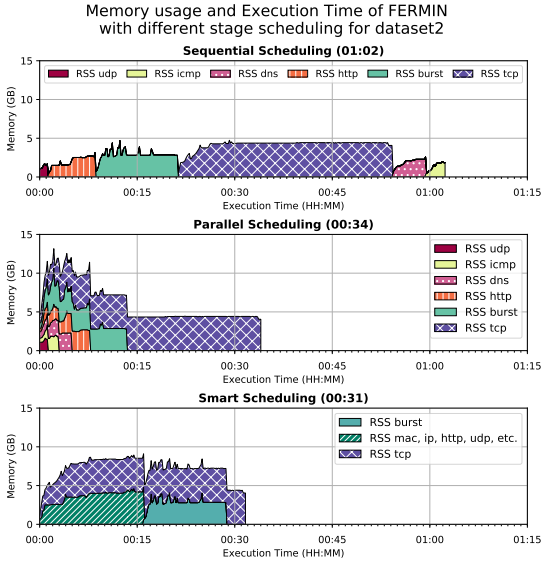
### 4.6.2. Una planificación mejor

Mediante las averiguaciones previas, se realizaron varias pruebas con cada conjunto de datos. En primer lugar, se utilizó una planificación en la cual todas las etapas se ejecutaron en paralelo. Tras ello, se probó una planificación *inteligente ad-hoc* en la cual se reúnen las etapas que requieren pocos recursos en un proceso y se encolan las tareas largas (tales como las etapas de análisis de ráfagas o TCP) en otro proceso, siendo estos procesos ejecutados en paralelo. Los resultados pueden ser comparados contra la planificación secuencial en la figura 4.8.

Como puede observarse en la figura 4.8, el tiempo de ejecución se ve reducido a la mitad en las planificaciones paralela e *inteligente*. Sin embargo, el uso de memoria resulta muy intenso en la planificación paralela, mientras que la planificación *inteligente* ofrece un buen compromiso en cuanto al tiempo de ejecución y el uso de memoria con mejores resultados en ambos aspectos. El mismo comportamiento puede ser observado en los dos conjuntos de datos.




Conjunto de datos: Dataset 1




Conjunto de datos: Dataset 2

Figura 4.8: Uso de memoria y tiempo de ejecución para diferentes planificaciones de las etapas de análisis. Las etapas demasiado cortas no se muestran.

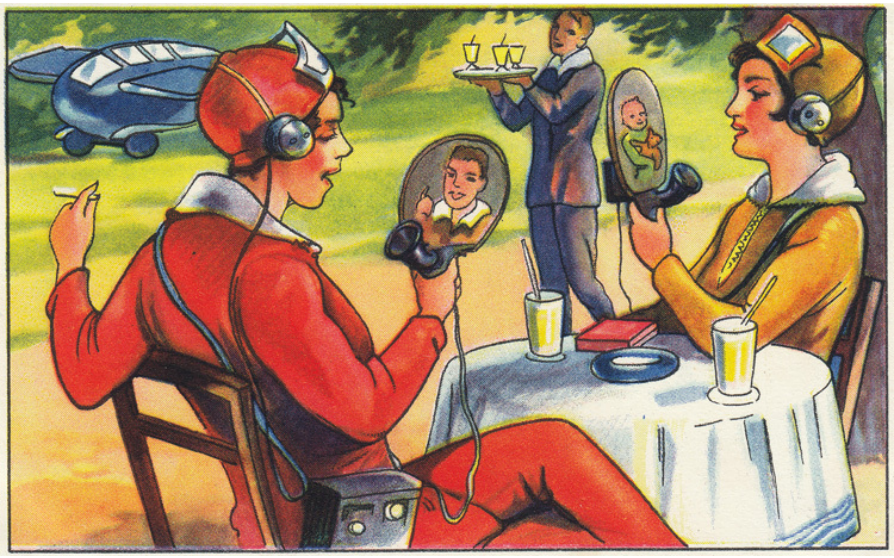
## 4.7. Conclusiones

En esta investigación se ha mostrado que el análisis macroscópico y proactivo del tráfico de red resulta esencial para la configuración adecuada de los sistemas reactivos de alarmas y monitorización de los grandes centros de datos, ayudando a dirigir el análisis microscópico hacia allí donde resulte productivo. Se han descrito y abordado los diferentes retos y requisitos del análisis proactivo, y destacado su importancia para la mejora de los sistemas de monitorización en tiempo real.

Como se ha visto en la sección de evaluación, la propuesta *FERMIN* ha excedido con creces el objetivo inicial de ser capaz de procesar volúmenes significativos de forma puntual y rápida. Con este fin, se han utilizado lenguajes de alto nivel ejecutados en *hardware* convencional, que proveen una amplia funcionalidad y versatilidad para la detección de comportamientos anómalos, que puedan requerir acciones correctivas.

Como trabajo futuro se planea extender la funcionalidad del sistema para mejorar su rendimiento, en busca de nuevas herramientas interactivas que permitan extender los resultados del informe a otros sistemas. Se trabaja además en la adición de nuevas fuentes de datos de otros protocolos, o registros de sistemas tales como Apache, etc.

Este trabajo está actualmente en proceso de envío en forma de artículo para revista bajo el título *Let the IT manager take a break: automatic traffic analysis from a hands-on perspective*. Así mismo, también se ha considerado como ejemplo de uso en el artículo *Diluting the Scalability Boundaries: Exploring the Use of Dissaggregated Architectures for High-Level Network Data Analysis*, realizado también por el autor de esta tesis, el cual será presentado en las conferencia *IEEE International Conference on High Performance Computing and Communications* el próximo Diciembre de 2017 en Bangkok, Tailandia, en representación del proyecto europeo *Disaggregated Recursive Datacentre-in-a-Box* (**dReDBox**).



Fantasías futuras, del álbum No. 3 de la fábrica Wagner, Series 12 y 13. 1925
Se lee: Teléfono privado inalámbrico y TV (*Drahtloses Privattelefon und Fernseher.*).
*Cada uno de ellos ahora tiene su propio transceptor y puede comunicarse con amigos y familiares en una cierta onda. Pero la tecnología de la televisión también se ha vuelto tan avanzada que puedes mirar a tus amigos cara a cara al mismo tiempo. El transceptor no está atado a la ubicación, sino que se lleva en una caja del tamaño de una cámara.*

CAPÍTULO 5

# DILUYENDO LAS BARRERAS DE LA ESCALABILIDAD

## EXPLORANDO EL USO DE ARQUITECTURAS DESAGREGADAS PARA EL ANÁLISIS A ALTO NIVEL DE DATOS DE REDES

Los centros de datos tradicionales están diseñados mediante una arquitectura rígida compuesta de servidores de diseño *ad-hoc* (*fit-for-purpose*) que aprovisionan recursos más allá de la carga media con el fin de poder enfrentar picos ocasionales de datos. Los centros de datos heterogéneos están impulsando hacia nuevas arquitecturas más rentables con mejor aprovisionamientos de recursos. Además, durante esta investigación se ha estudiado la viabilidad para utilizar arquitecturas desagregadas para aplicaciones con un uso intensivo de datos, en contraste con el enfoque monolítico de las arquitecturas orientadas a servidores. Estas arquitecturas han demostrado ser mejores en cuanto a la proporcionalidad de los recursos aprovisionados, así como el consumo eléctrico y el escalado tanto horizontal como vertical. En concreto, se han realizado pruebas mediante un sistema de análisis proactivo de tráfico, *FERMIN* presentado en el capítulo anterior, en el cual, la demanda de recursos resulta altamente variable. En este trabajo se pone de manifiesto que las arquitecturas desagregadas pueden ayudar a aliviar las desigualdades en la carga de trabajo de los centros de datos, ofreciendo una mayor elasticidad en los recursos asignados. Esta investigación también se centra en los actuales puntos de vista y retos de la arquitectura Disaggregated Recursive Datacentre-in-a-Box (*dReDBox*).

## 5.1. Introducción a las arquitecturas desagregadas

Hoy en día los grandes centros de datos sirven multitud de aplicaciones diferentes, muy frecuentemente a través de máquinas virtuales que ofrecen un mayor nivel de abstracción. Tradicionalmente, los servidores de los centros de datos han sido construidos en base a bloques de construcción monolíticos y rígidos. Esto es, placas base con un número fijo de procesadores y bancos de memoria. Estos bloques de construcción definen las características de las máquinas virtuales que serán ejecutadas en los centros de datos. Las aplicaciones deberán a su vez adaptarse a las características de estas máquinas virtuales, y el escalado de la aplicación es normalmente obtenido mediante la adición de nuevas instancias de estas máquinas virtuales.

La cuestión es si esta arquitectura rígida, basada en bloques de construcción monolíticos, va a ser capaz de satisfacer los requisitos de los centros de datos del futuro. Próximamente, se espera un incremento significativo tanto del tamaño de los centros de datos [82, Fig. 4] [40, Fig. 2] como del volumen de datos [39, Fig. 5.8] [40, Fig. 4]. Adicionalmente, los próximos avances tales como el Internet de las Cosas [83] (en inglés, *Internet of Things*, abreviado IoT) y las redes de 100 Gbps [85, pg. 7] [86] supondrán nuevos retos para los futuros centros de datos.

En realidad, la rigidez impuesta por los bloques de construcción monolíticos dibuja una frontera clara en como los recursos de computación, memoria, almacenamiento y redes podrán ser expandidos durante futuras actualizaciones de una arquitectura de un centro de datos. Las decisiones tomadas durante la fase de diseño condicionarán la forma en que un sistema evoluciona, con un impacto directo en la utilización eficiente de los recursos, actualizaciones costosas y la eficiencia energética.

Recientemente, las arquitecturas desagregadas han sido propuestas como alternativa para superar la rigidez de los servidores convencionales. Los beneficios de la desagregación han sido ampliamente discutidos en la literatura [87], ya sea a través de la mejora del escalado vertical, proponiendo *blades* (tarjetas) de memoria independientes que desagreguen los recursos de memoria [88][89]; o a través del estudio de las capacidades de las redes para la desagregación de recursos [90][91] en centros de datos. Las interconexiones ópticas han sido también abordadas en la literatura [92] con el fin de mejorar tanto el consumo energético en las interconexiones *intra-rack* como a través de los conmutadores (*switch*) ópticos *Top-of-Rack* (ToR) y las comunicaciones *inter-rack* con nuevas arquitecturas de conmutadores ópticos [93].

Además, la distrubución de recursos en arquitecturas orientadas a servidores tiene un claro impacto en el consumo energético, lo cual ha sido abordado durante los ultimos años "reduciendo el consumo durante los periodos ociosos o de baja utilización" (*"by reducing power draw during idle periods or when at low utilization"*) [82, pg. 24]. Las arquitecturas desagregadas contribuyen a esta tendencia ofreciendo un control de grano fino sobre el aprovisionamiento de recursos y su utilización. Además del mal aprovisionamiento de recursos espacial, las cargas altamente variables en el tiempo (v. gr. tareas diurnas y nocturnas) producen un consumo desigual de los recursos disponibles.

No obstante, el costo-beneficio todavía supone un caso de estudio [94] y resulta muy difícil de cuantificar en la etapa actual. Sin embargo, se pone de manifiesto que los ahorros en coste pueden llegar a suponer una de las mayores ventajas de este tipo de arquitecturas debido a su diseño flexible y sus componentes de bajo consumo.

Con este fin, el proyecto *dReDBox* [1] (*Disaggregated Recursive Datacentre-in-a-Box*) aceptó el reto [104] de romper las barreras de los servidores en busca de la materialización del concepto de desagregación, beneficiándose de las mejoras tecnológicas de la interconexión de los componentes, como los conmutadores totalmente ópticos de baja latencia [95] [96]. La idea principal de la arquitectura *dReDBox* consiste en diluir la unidad base de los centros de datos mediante un núcleo de material optoelectrónico de alta velocidad, baja latencia que aúne componentes físicamente distantes en términos de ancho de banda y latencia. Además, *dReDBox* propone una arquitectura adaptable de bajo consumo para centros de datos, diferenciándose del paradigma de placa base como unidad base del servidor hacia un esquema con bloques flexibles definidos por *software*.

Como se ha indicado anteriormente, existen claros efectos colaterales en la forma en que los centros de datos están construidos hoy en día. Algunas de estas limitaciones son las siguientes:

- La proporcionalidad de los recursos en los bloques de construcción de los centros de datos afectan a las inversiones de capital (*Capital Expenditure*, CAPEX) y los costes operacionales (*Operational Expense*, OPEX) durante los ciclos de actualización. Por ejemplo, Han et al. destacan a partir de trazas de un *clúster* de Google [97], la gran disparidad en la relación de uso almacenamiento/memoria con el uso de CPU en sus infraestructuras de centros de datos [90].

- La reserva de recursos en máquinas virtuales está atada a los recursos disponibles en la máquina anfitrión, produciendo una fragmentación y baja utilización de

[1] `http://www.dReDBox.eu/`

los recursos de memoria, cambiando drásticamente en el tiempo [88, Fig. 1]. Por ejemplo, Samih destaca que en un *clúster* con una memoria agregada de 437 TBytes tan sólo el 69 % está siendo usada, con 68 TBytes en dispositivos *swap*, mostrando como algunos servidores no tienen recursos suficientes mientras otros tienen memoria libre que podrían estar compartiendo [98, Fig. 1].

- De la misma manera, actualizar o incorporar una nueva tecnología puede resultar tedioso y perjudicial para la ejecución normal del servidor. Los costes de actualización y métodos de aprovisionamiento de recursos han sido ampliamente estudiados en la literatura [99].

Los puntos 1 y 2 suponen serios obstáculos para sostener una alta eficiencia, ya que por lo general los centros de datos, separan aplicaciones autónomas (*standalone*) que se ejecutan en servidores individuales y aplicaciones distribuidas que requieren varios servidores para ser desplegados, estresando las ineficacias de los recursos. La primera tiene la ventaja de aportar escalado vertical mientras que el segundo se beneficia de la distribución horizontal de su carga de trabajo. Sin embargo, ambas son propensas a fragmentar los recursos disponibles durante los momentos ociosos, lo cual no tiene por qué coincidir con los periodos de máxima utilización del resto de aplicaciones. Todo esto motiva aun más la necesidad de arquitecturas desagregadas.

Durante el desarrollo del proyecto *dReDBox*, se han estudiado diferentes casos de uso [100], representativos de una gran clase de posibles aplicaciones que el sistema podría hospedar en producción. La siguiente sección se centrará en el problema del análisis de datos y en cómo las arquitecturas desagregadas pueden adaptarse a sus necesidades.

## 5.2. Análisis de datos en arquitecturas desagregadas

Tareas intensivas como minería de datos, procesamiento de señal o sistemas de datos masivos tienen una demanda altamente variable tanto de recursos espaciales como temporales que usualmente requieren exprimir bien el escalado vertical, bien el horizontal, o inclsuo los dos.

Por ejemplo, indexar una cantidad masiva de documentos en bases de datos NoSQL, como Elasticsearch [2] o Apache Solr [3], puede requerir un gran uso de memoria para

[2] `https://elastic.co`
[3] `http://lucene.apache.org/solr/`

aprovisionar y encolar los documentos, mientras que, por otro lado, las operaciones de agregación (v. gr. calcular percentiles de intervalos sobre un gran intervalo de tiempo y documentos) en estas plataformas de búsqueda causan una alta demanda de CPU. Por lo tanto, la fragmentación de recursos surge bajo cargas computacionales variables y heterogéneas como las mencionadas anteriormente.

Otro ejemplo de esta demanda variable es el tráfico de red en los centros de datos, en el cual, la cantidad de flujos IP, transacciones HTTP, y similares, varían enormemente en el tiempo y entre estaciones. En un gran centro de datos, un día de tráfico puede rondar los 2-3 TBytes. La figura 5.1 muestra el volumen de registros por TB de traza de tráfico obtenidos a través de la disección [32][77] de diferentes capas y protocolos. Nótese que el volumen de los registros diseccionados es significativo, lo cual requiere una planificación cuidadosa de la capacidad del sistema, en particular la memoria de la máquina para realizar análisis de la información extraída.

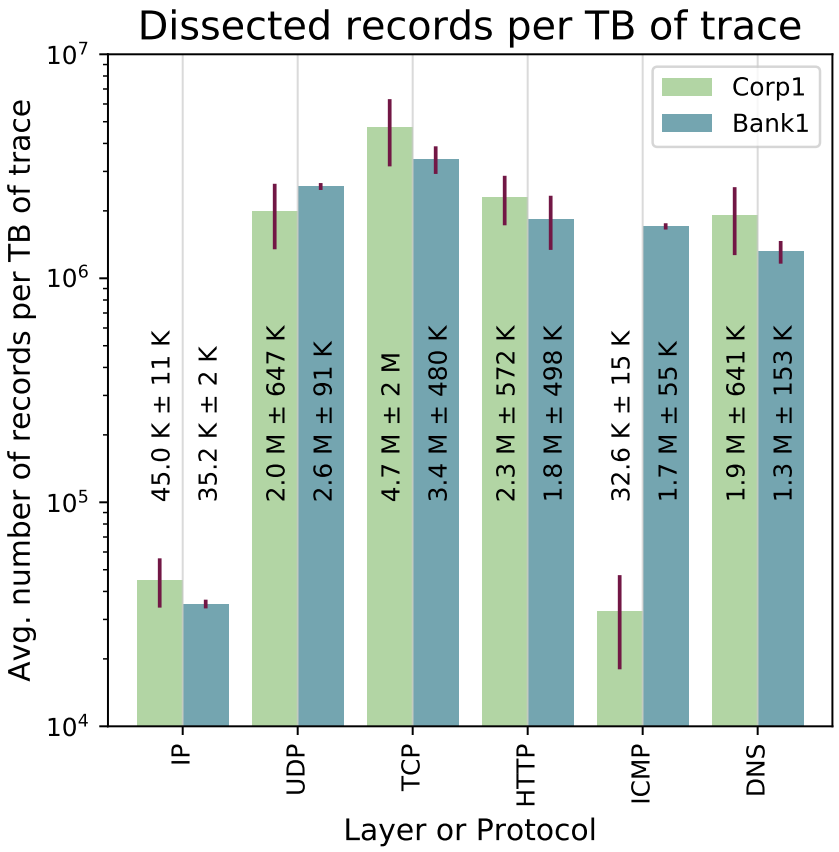



Figura 5.1: Alta variación en el nº de registros por TByte de traza en diferentes redes empresariales, obtenidos mediante disectores de distintos protocolos y capas de tráfico.

Por tanto, el escalado vertical de los recursos es crucial si se propone evitar el coste de la distribución de datos una vez estos ya han sido centralizados en el centro de datos. Además, en muchos casos, como ya se ha advertido en capítulos anteriores, la naturaleza de la información supone un reto para una distribución equitativa de los datos. De nuevo, como concluyen Shi et al. respecto a los flujos IP: *"Bajo ciertas distribuciones de tipo Zipf de los tamaños de flujo, no es suficiente utilizar un sistema de hashing para balancear la carga."* [76] (*"under certain Zipf-like flow-size distributions, hashing alone is not able to balance workload."*).

En esta investigación se ha estudiado la viabilidad del uso de arquitecturas desagregadas para tareas de análisis intensivo de datos, estudiando un caso de uso relacionado con el análisis del tráfico. Como se ha indicado previamente, la desagregación generalmente tiene una penalización en el rendimiento en cuanto a las latencias de los recursos remotos cuando el ratio de operaciones no locales se incrementa. Sin embargo, las penalizaciones de rendimiento en arquitecturas desagregadas todavía son sujeto de estudio [101] y son fuertemente dependientes de la tarea realizada y sus patrones de acceso de entrada/salida.

Por tanto, en los centros de datos tradicionales, la forma y características de los datos requieren el diseño de soluciones *ad-hoc* por encima de las necesidades medias con el fin de ser capaces de acometer los picos de carga ocasionales. Es decir, aprovisionar más recursos verticales u horizontales de lo necesario usualmente, sino de ambos.

Particularmente, las arquitecturas desagregadas son ideales para aliviar estos picos de carga ocasionales en las tareas de análisis proactivas. Por ejemplo, las ventas en las grandes superficies de los Estados Unidos de América durante la temporada de ventas de Navidad registran un incremento de un 40% respecto al mes anterior [102] [103] lo cual podría requerir una mayor cantidad de recursos para el análisis de las transacciones comerciales comparado con el resto del año. Este pico estacional de carga puede no merecer el coste de tener recursos infrautilizados durante el resto del año. Sin embargo, en un hipotético centro de datos desagregado, las variaciones de carga serían aliviadas, asignando recursos remotos ociosos para cubrir las demandas excepcionales de recursos.

En base a lo anterior, el resto del capítulo estará estructurado de la siguiente manera. En primer lugar se presentará la arquitectura *dReDBox* para centros de datos, seguido de la descripción de un simulador de arquitecturas desarrollado en el *Centro Nacional de Supercomputación* (*Barcelona Supercomputing Center*, BCS-CNS) como banco de pruebas para las consiguientes evaluaciones del caso de uso presentado en la sección 5.5, en el cual, se profundizará en el caso particular de análisis de datos a alto nivel en

grandes redes. Finalmente se resumirán las conclusiones de la evaluación realizada y las futuras expectativas en el área.

## 5.3. La arquitectura dReDBox

Para abordar los retos de las arquitecturas desagregadas, el proyecto dReDBox [104] adopta una arquitectura vertical, cuyo nivel más bajo consiste en un sistema de interconexión para la comunicación de la memoria remota, el cual aprovecha al máximo las soluciones ópticas disponibles para latencias en el orden de decenas de nanosegundos. Con la interconexión de los controladores y módulos de memoria remota mediante novedosas redes ópticas escalables se obtienen anchos de banda de bisección del orden de varios Terabit por segundo.

Entre las muchas innovaciones previstas del proyecto, destacan particularmente:

- Un nuevo monitor de máquinas virtuales (*hyper-visor*) distribuido para compartir recursos, que permita la ejecución de máquinas virtuales convencionales, el cual sería adoptado como el contenedor de ejecución.
- Uso de un *software* para controlar los recursos a nivel de programación *hardware*. Este *software* de orquestación sería conectado con los recursos aprovisionados de mayor nivel, el administrador y los sistemas de planificación mediante una API.
- La reducción del consumo de energía en todas las capas. Las plataformas *hardware* propuestas proveen una interfaz IPMIv2 por cada componente, ofreciendo un control total para la orquestación desde el *hyper-visor* (v. gr. apagar los componentes ociosos).

La arquitectura dReDBox permite que el uso de recursos fluya entre las unidades básicas de la misma manera que las máquinas huésped proveen flexibilidad de recursos a los procesos hospedados en las máquinas virtuales, reservando porciones de *hardware* dentro de un servidor.

### 5.3.1. Arquitectura del servidor y el *rack*

La plataforma *dReDBox* se relaciona con diferentes bloques de *hardware* mediante adaptadores de memoria remota (*Remote Memory Adapters*, RMA), con SoC (*Systems on a Chip*) de alto rendimiento para los bloques de computación, todo ello integrados con

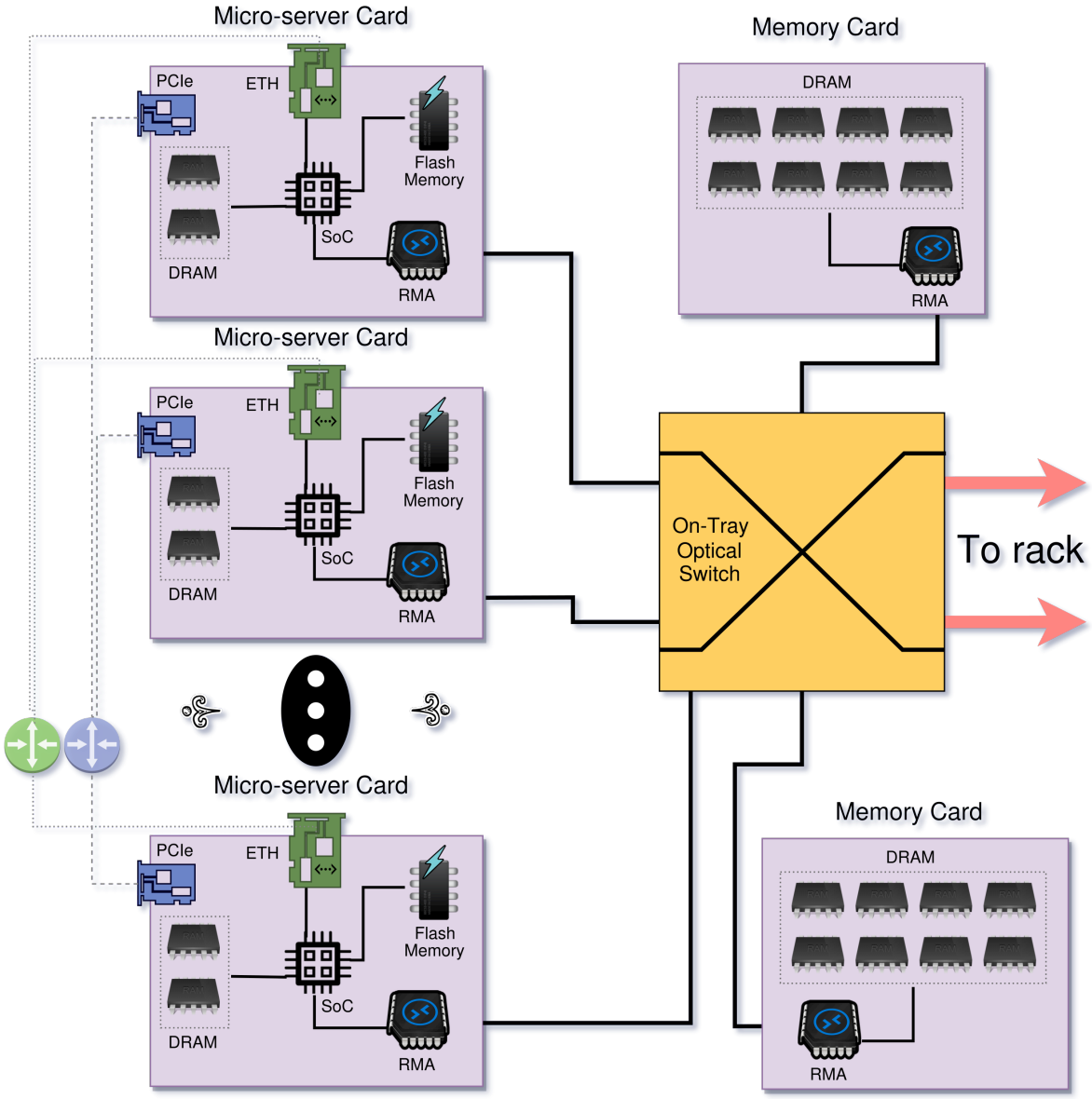



Figura 5.2: Abstracción de un tipo de tarjeta servidor (*server tray*). ETH abrevia *Ethernet* y RMA *Remote Memory Adapters*.

un RMA local, memoria local, memoria *flash* y un BMC (*Board Management Controller*) basado en ethernet. Cada uno de estos bloques está basado en micro-servidores, haciendo uso de placas base de pequeño factor.

La arquitectura elegida para el sistema de los SoC está basada en procesadores ARMv8-A Cortex 64-bit [105] debido a su bajo consumo energético y coste reducido.

A pesar de que los procesadores de aplicaciones basados en ARM pueden ser encontrados en el 85% de los dispositivos móviles [106], recientemente ha habido un gran interés por el uso de núcleos ARM también en la computación de alto rendimiento [107] [108]. ARM espera que *"El 25% de los servidores despachados en 2020 usen chips basados en ARM"* (*"25% share of servers shipped in 2020 will be using ARM-based chips."*) [106].

La figura 5.2 representa una abstracción a alto nivel de la actual arquitectura, mostrando como los diferentes componentes se interconectas entre ellos dentro de una tarjeta placa base genérica de *dReDBox*. A nivel *rack* varias tarjetas se interconectan entre ellas a través de un switch ToR (*Top of rack*).

### 5.3.2. Interconexión conmutada electro-óptica

La desagregación depende inherentemente del rendimiento de la red, el cual es crucial para servir recursos remotos. Es clave obtener un alto nivel de conectividad, granularidad del ancho de banda y bajas latencias para el desarrollo de estas redes.

Por ejemplo, las densas interfaces transmisor-receptor opto-electrónicas han sido consideradas como MBO (*Mid Board Optics*) [110] debido a su bajo consumo y alto ancho de banda. Muchas variables, como la banda de transmisión en la que se opera, deben ser tenidas en cuenta para ofrecer el máximo rendimiento.

Con respecto a la interconexión de la memoria, esta siempre impone unas restricciones muy altas de memoria, la arquitectura *dReDBox* considera varias opciones de conmutación, incluyendo conmutación de circuitos y de paquetes totalmente óptica, explorando también arquitecturas híbridas de conmutación. El número de extremos conectados se incrementa debido al escaneo de recursos continuo (*resource pooling*), requiriendo sistemas de conmutación a todos los niveles (en la tarjeta, entre-*racks*, intra-*rack*). Esto se acentua cuando los micro-servidores son la unidad base de computación.

### 5.3.3. Desagregación de la memoria

El proyecto *dReDBox* desagrega la memoria colocando módulos en una tarjeta dedicada de memoria e interconectándolos por el sistema y con los adaptadores remotos

de memoria. El desarrollo de interfaces de memoria y su incorporación lógica para la transmisión a través de la red óptica es clave, y *dReDBox* integra SoC existentes y componentes de memoria remota fáciles de comunicar con un el sistema de bus de datos de un procesador remoto. Este componente soporta ser configurado mediante Entrada/Salida mapeada en memoria haciendo uso de rangos especiales de memoria. La memoria local convencional está también disponible con el fin de soportar los procesos del sistema.

### 5.3.4. Soporte para la desagregación en el sistema operativo

La plataforma *dReDBox* soporta máquinas virtuales como la unidad básica de ejecución, ofreciendo la ejecución personalizable de máquinas virtuales convencionales para las aplicaciones, sin comprometer su rendimiento. Por tanto, aplicaciones, herramientas o sistemas desarrollados para ser ejecutados en *hardware* convencional podrán ser desplegadas en la plataforma desagregada sin modificaciones.

El hyper-visor de la plataforma está basado en *Kernel-based Virtual Machine* (KVM) [4], un módulo del núcleo que permite que un sistema operativo Linux hospede varias máquinas virtuales. Los sistemas anfitriones en cada micro-servidor *dReDBox* pueden no ser capaces de detectar todos los componentes disponibles en la plataforma durante la inicialización del *hardware* local a través de la BIOS. En realidad, la BIOS de los micro-servidores provee únicamente información acerca de los componentes conectados físicamente. Por lo tanto, durante la inicialización, el sistema anfitrión deberá recabar esta información mediante las herramientas de orquestación.

### 5.3.5. Reserva y orquestación de recursos

La orquestación en la plataforma *dReDBox* es clave para una reserva eficiente de los recursos del centro de datos. La redirección de información a través de la red de conmutación es esencial para la interconexión de cualquier combinación de componentes. El enfoque de *dReDBox* provee un espacio de direcciones de memoria disponible a través de todo el centro de datos, manteniendo una distribución coherente de ello, la cual soporta segmentación y *memory ballooning*. También, el control de cada componente IPMIv2 ofrece una reducción potencial en el consumo de energía. Un API estandarizado integra la capa orquestación junto con las herramientas de administración de recursos.

[4] `https://www.linux-kvm.org/`

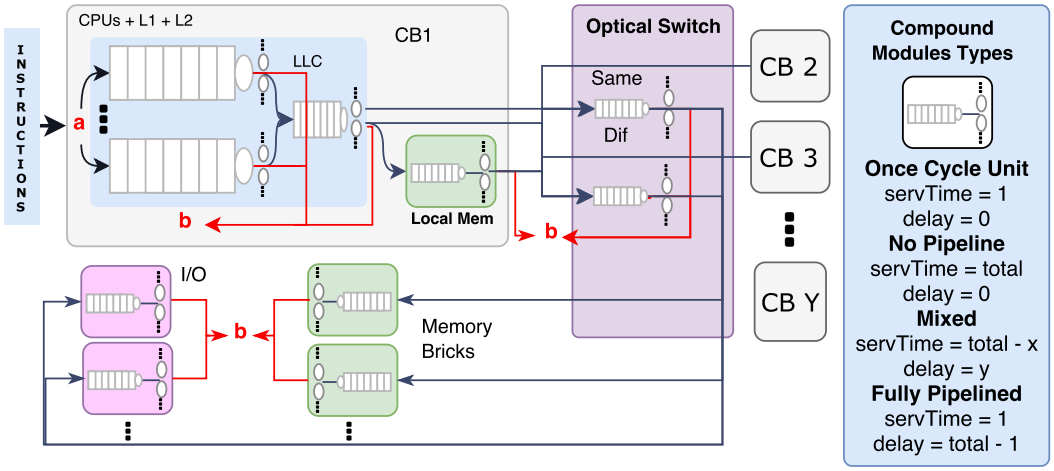



Figura 5.3: Modelo iQ con los diferentes componentes de la infraestructura dReDBox.

Con todo esto en mente, antes del despliegue de una máquinas virtual, será necesario realizar un proceso de planificación de recursos y síntesis de la plataforma para reservar los recursos necesarios y preparar la plataforma de interconexión de una forma eficiente.

## 5.4. Simulando una arquitectura desagregada

Previo a la discusión del caso de uso propuesto, en esta sección se describe el proceso de simulación usado para la evaluación de los sistemas en la arquitectura dReDBox. Este simulador es empleado para modelar el comportamiento de una arquitectura desagregada basándose en modelos de colas. El modelo iQ [109] desarrollado en el *Barcelona Supercomputing Center* ha sido diseñado para representar y analizar la desagregación de memoria. Un simulador estadístico del sistema completo basado en colas fue desarrollado para analizar el rendimiento de las aplicaciones en sistemas desagregados de una forma rápida y precisa. Estos modelos emplean estructuras de colas, reenvío de mensajes, y acumulación de latencias que permiten modelar tales sistemas.

Particularmente, este proceso comienza cuando un mensaje es recibido al final de una estructura de colas. Entonces, este mensaje es propagado hasta que alcanza la cabeza de la cola. Finalmente, se añade un retraso para emular la cantidad de tiempo que la acción hubiera tardado en realizarse para ese mensaje en concreto. Por ejemplo, en el caso de una ALU de enteros, la cola modelaría la cola de entrada de la ALU donde el mensaje representaría una instrucción aritmética en particular (v. gr. adición, sustracción, igualdad, etc.). En consecuencia, el retardo añadido dependerá de la cantidad de tiempo que la unidad aritmético-lógica tarde en ejecutar esa instrucción aritmética.

Cuadro 5.1: Latencias de la desagregación de memoria.

| Nombre del módulo | Latencias actuales (ns) | Número de accesos por petición |
|---|---|---|
| FPGA Address Translation | 72 | 1 (CB) |
| Ingress/Egress | 6.25 | 4 (2 CB, 2 MB) |
| Network-on-Chip | 22.4 | 4 (2 CB, 2 MB) |
| PCS/PMA | 251 | 4 (2 CB, 2 MB) |
| DDR4 | 62.5 | 1 (MB) |

Por tanto, el proceso de simulación ocurre como una colección discreta de eventos, los cuales en este modelo son representados como un ciclo computacional. El progreso de la ejecución (incluyendo la generación de instrucciones, su propagación, esperas, ejecución, y retirada así como el uso de recursos, saltos de ramas y fallos de memoria) es simulado dentro del modelo de colas para cada evento (v. gr. cada ciclo). El rendimiento total es medido como una colección de mensajes procesados por eventos totales, v. gr. Instrucciones por Ciclo (IPC). El modelo de colas dirigido por eventos hace uso de una combinación modular de varias estructuras de colas, rastreo de dependencias, y ejecución probabilística de flujos para simular sistemas concretos.

La metodología basada en modelos de colas emula los componentes del procesador abstrayendo los detalles de su implementación dentro de los componentes modulares, compuestos de estructuras de colas, parámetros de retardos, y la generación probabilística de mensajes y control de eventos.

La figura 5.3 muestra los bloques de interconexión a través de conmutadores ópticos por motivos de simplificación. Con el fin de representar los procesadores, memorias y otros componentes usando modelos de colas, se ha implementado una estructura de colas modular que modela los distintos comportamientos a través de un conjunto de configuraciones de variables (la letra **a** roja en la figura 5.3 indica el comienzo del procesamiento de instrucciones). Abajo a la izquierda de la figura 5.3 se representa el módulo utilizado para representar cada componente. Este módulo está formado por una cola, un servidor y un retardo. La longitud de la cola y los retardos requeridos para procesar las instrucciones son flexibles y configurables. El parámetro de longitud es utilizado para modelar la contención de recursos y su disponibilidad. El tiempo de servicio (*servTime*) representa el tiempo necesario para procesar una instrucción hasta que la siguiente instrucción puede empezar a ser procesada. El tiempo total de ejecución de la instrucción dentro de un módulo será la suma de sus propios *servTime* más el

tiempo de servicio de todas las instrucciones ejecutadas previamente (encadenamiento, *pipelining*). Cuanto menor tiempo de servicio, mayor nivel de encadenamiento y viceversa. El parámetro de retardo (*delay*) es empleado para complementar el *servTime* y asegurar que el retardo total correspondiente es añadido a la instrucción.

Las instrucciones son generadas de acuerdo a la información estadística recolectada durante la etapa de análisis de rendimiento (*profiling*) usando la herramienta para Linux, perf [5] y el *pintool* mica [6]. Estas instrucciones son introducidas en el punto de entrada de los bloques de computación como se muestra en la figura 5.3. Después, dependiendo de los valores de probabilidad, las instrucciones se moverán de una cola a otra o al sumidero (punto **b** en la figura 5.3). Las instrucciones se mueven entre los diferentes niveles de caché y de memoria local. En el caso de que las instrucciones necesiten acceder a diferentes bloques remotos (E/S, Memoria, etc.), estos podrían requerir ser enviados a través de conmutadores de circuitos ópticos (*Optical Circuit Switch*) (OCS).

Por ejemplo, en [109, Tabla 2] Meyer et al. validan el simulador mediante una comparación entre diferentes escenarios entre el valor IPC real y el simulado. Aunque estos resultados corresponden a evaluaciones pasadas que emplean modelos previos, sirven como guía. El IPC se usa como indicador de rendimiento en los experimentos, ayudando a analizar el impacto de la desagregación en diferentes casos de uso.

Específicamente, durante la evaluación, se simuló el procesador Intel® Xeon®E5-2630 0 @ 2.30GHz, incluyendo todas sus ALUs y niveles de caché. Los valores utilizados para simular la desagregación son presentados en la tabla 5.1. Esta tabla también incluye el número de veces que un módulo es accedido por cada petición de memoria remota en el bloque de computación (CB) y en los bloques de memoria (MB). Los valores de latencia presentados en la columna "Latencias actuales"de la tabla 5.1 son resultados preeliminares que corresponden al prototipo actual de *dReDBox* actualmente en desarrollo. Estas latencias así como la degradación del rendimiento serán reducidas durante la fase de refinamiento del proyecto. Se espera que estas latencias se reduzcan a la mitad tras esta fase.

[5] `https://perf.wiki.kernel.org/`
[6] `http://kejo.be/ELIS/mica/`

## 5.5. Análisis de red a alto nivel

Con el fin de evaluar la viabilidad de usar arquitecturas distribuidas para tareas de análisis con uso intensivo de datos, se considera el siguiente caso de uso de análisis de infraestructuras. Actualmente, la monitorización y auditoría de redes resultan un problema verdaderamente difícil de escalar cuando se enfrentan velocidades de las espinas dorsales empresariales (v. gr. entre 10 y 100 Gbps) [31] [86].

Hoy en día, la plataforma de analíticas trata de aprovechar al máximo las capacidades de los sistemas convencionales, ya que migrar a otras placas base con más recursos supone el coste prohibitivo de mover los datos a un lugar diferente. También, debido a la naturaleza del tráfico, la distribución de la captura y el proceso de análisis supone un gran reto. Esto es por lo que las arquitecturas escalables y desagregadas como la concebida en el proyecto *dReDBox* se adaptan a las necesidades de la monitorización de red y los sistemas de auditoría.

Para nuestro ejemplo particular se consideró la solución de análisis de tráfico, *FERMIN* [7](*Factual Executive Report of a Monitored IP Network*) expuesta en el capítulo anterior, la cual está orientada a la generación de informes automáticos con el fin de mejorar la monitorización del tráfico en grandes infraestructuras de TI.

Como se explicó en capítulos anteriores, el escenario típico de despliegue considera una gran infraestructura de TI que realiza mantenimiento continuo de servicio (ITSCM en sus siglas en inglés), realizando en consecuencia análisis de riesgo para cada servicio de TI con el fin de señalar las vulnerabilidades, activos y amenazas y contramedidas de

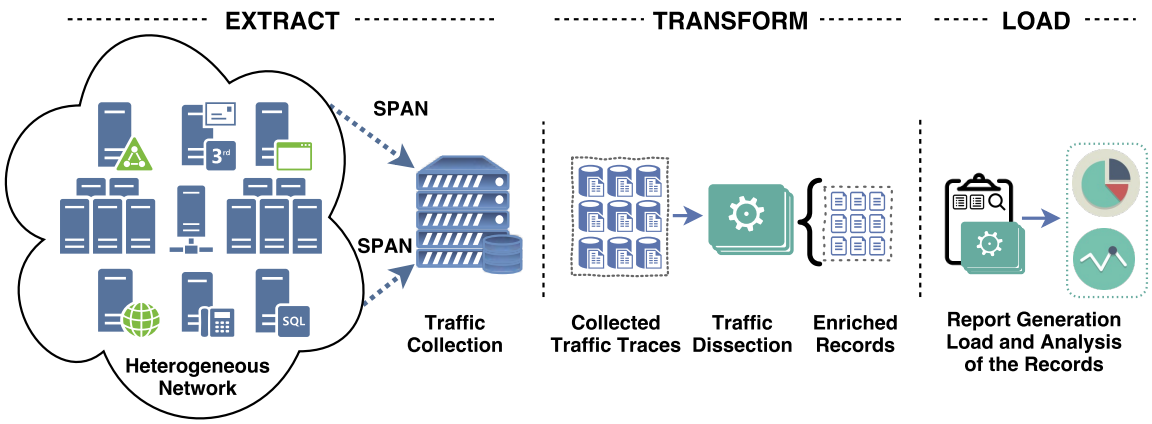



Figura 5.4: Diferentes etapas del proceso de análisis de tráfico a alto nivel.

[7] http://www.naudit.es/fermin

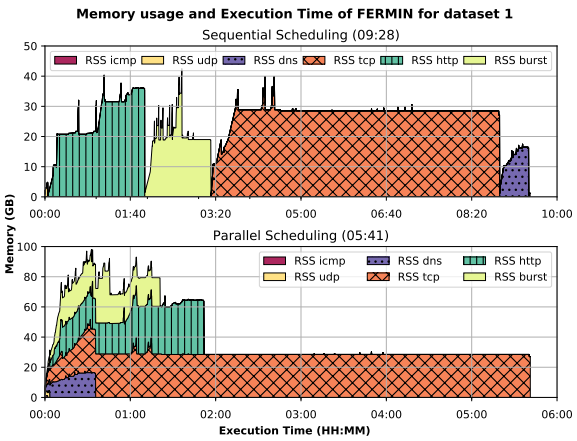
cada uno de estos servicios. Como parte de este proceso, se realiza un análisis del tráfico ya sea diariamente o semanalmente durante la noche, mediante un proceso automatizado que genera un informe del tráfico analizado. Posteriormente, las contramedidas y acciones correctivas se despliegan en los distintos sistemas de alarmas y monitorización. La figura 5.4 resume todo este proceso.

Debido a que ya se han descrito los detalles particulares de *FERMIN* en el capítulo anterior, no es el propósito de este capítulo insistir en estos aspectos.

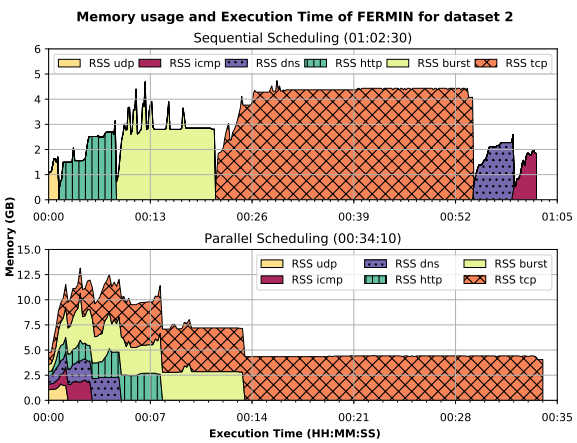
A pesar de que las redes muestran unos comportamientos claramente diferentes (v. gr. día-noche o trabajo-vacaciones) durante distintas épocas del calendario, existen eventos inesperados que podrían alterar el tráfico significativamente, y por tanto, la carga computacional de un sistema de análisis de red puede ser impredecible.

Por ejemplo, en la figura 5.5 se muestran dos casos de carga de trabajo distintos, correspondientes a un análisis proactivo realizado en sondas de red tradicionales mediante Intel®Xeon E5-2640 v3 @ 2.6Ghz con 128GB de memoria RAM. Estas figuras ya fueron comentadas en la sección **“Metodologías para el análisis automatizado del tráfico”** del capítulo anterior, en la cual se describe el origen y composición de los distintos conjuntos de datos.

Como puede observarse, los recursos necesarios para abordar el caso de estrés llegan a ser incluso un orden de magnitd mayores que el caso medio. En realidad, con el fin de soportar con estos picos impredecibles de tráfico, se necesitaría aprovisionar muchos más




Caso de estrés con alta carga de recursos.




Caso medio de uso de recursos.

Figura 5.5: Ejecución etapa a etapa de un análisis proactivo de red en diferentes escenarios. Se muestra el tiempo de ejecución de cada etapa y su correspondiente uso de memoria. RSS indica el uso de memoria residente (*Resident Set Size*).

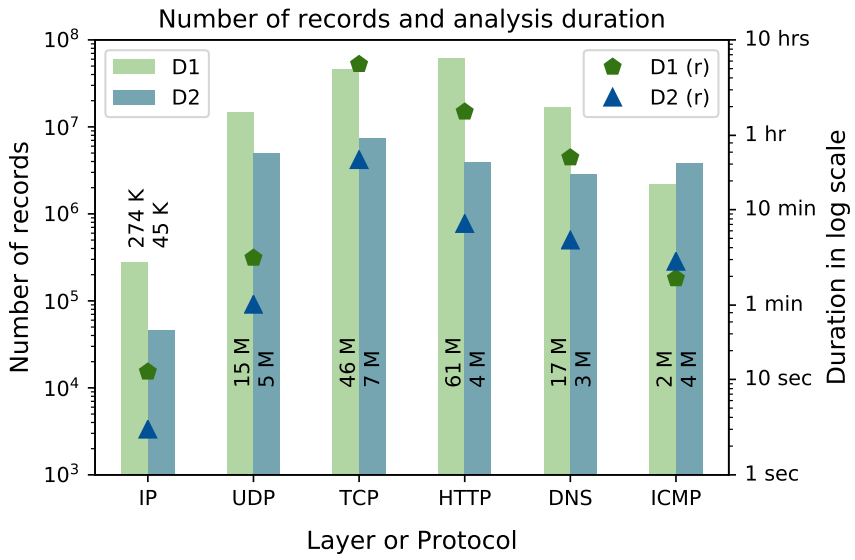



Figura 5.6: Cantidad de registros diseccionados por capa/protocolo junto con su correspondiente tiempo de procesamiento en FERMIN. D1 y D2 indican el conjunto de datos **Dataset 1** y el conjunto de datos **Dataset 2** respectivamente.

recursos de los que se necesitarían habitualmente. Durante las fechas pico, el principal problema no es sólo la variación del volumen de datos, sino el gran incremento en las conversaciones IP, flujos TCP/UDP, o las transacciones HTTP.

La figura 5.6 representa el gran incremento (y diferencia) en el número de registros diseccionado, correspondientes a las conversaciones IP, flujos TCP/UDP, transacciones HTTP, etc. entre el caso de estrés y el habitual, indicando también la duración de los análisis. En una arquitectura desagregada, los picos de carga serían aliviados haciendo uso de recursos remotos ociosos.

Cuadro 5.2: Resultados de la simulación

| Número de Extremos | IPC[1] con latencias actuales | Sobrecoste[2] [%] | IPC con latencias esperadas | Sobrecoste [%] |
|---|---|---|---|---|
| 1 | 0,25 | 81,75 | 0,456 | 66,72 |
| 2 | 0,26 | 81,02 | 0,463 | 66,20 |
| 4 | 0,29 | 78,83 | 0,469 | 65,77 |
| 8 | 0,30 | 78,10 | 0,465 | 66,06 |

[1] Instrucciones por Ciclo

[2] $Overhead = \frac{IPC_{sim} - IPC_{disagg}}{IPC_{sim}} * 100$

## 5.6. Evaluación, conclusiones y trabajo futuro

Ante lo dicho, se realizó una evaluación de *FERMIN* en la forma descrita en la sección 5.4, con el propósito de recolectar información estadística acerca de los patrones de acceso de memoria y las dependencias entre instrucciones. La simulación fue validada mediante esta información realizando una comparación entre el valor real y simulado de instrucciones por ciclo (IPC), reportando un error del 4.5% sobre el IPC real.

Después, la aplicación fue simulada, esta vez en el modelo desagregado, obteniendo los resultados expuestos en la tabla 5.2. El IPC simulado sin desagregación es de 1,37. La tabla presenta diferentes resultados para tarjetas de memoria con 1, 2, 4 u 8 extremos de comunicación, ofreciendo cada extremo unos 16 Gbps. Los resultados muestran que para las latencias actuales, la sobrecarga es de un 80%, lo cual puede ser reducido a un 66% si las latencias se reducen a la mitad, algo que se espera conseguir en futuras mejoras de la arquitectura dReDBox.

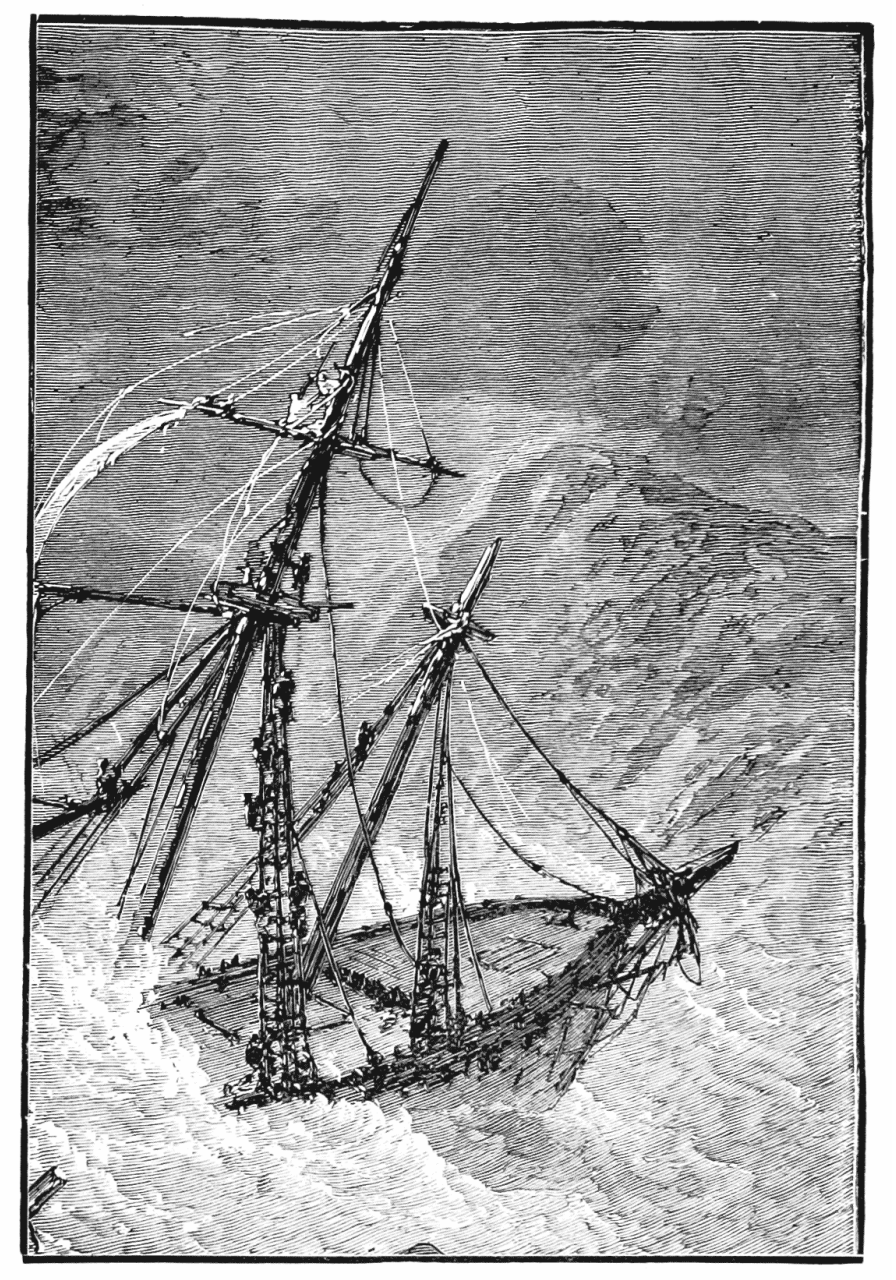
Es necesario advertir que el acceso y aprovisionamiento de los recursos de memoria remotos no deberían ser el comportamiento usual sino servir para aliviar situaciones de estrés como los descritos en secciones anteriores. En este sentido, el compromiso entre la sobrecarga de latencias y la proporcionalidad de los recursos aprovisionados podría merecer la pena para sistemas de procesamiento en lotes (*batch-processing*), como por ejemplo los sistemas de análisis proactivo de red, entre otros sistemas con gran variabilidad de carga. Estos sistemas no tienen restricciones de tiempo real estrictas y pueden postergar sus tareas de procesamiento en favor de un mejor aprovisionamiento y planificación de los recursos. En estos casos, los beneficios de las arquitecturas desagregadas, es que para las tareas de análisis de datos descritas anteriormente, que serían imposibles de realizar debido a sus altos requerimientos de memoria, podrían ser realizadas con una sobrecarga del 66% en su rendimiento.

Un beneficio adicional de realizar un aprovisionamiento remoto de los recursos de memoria es que el paralelismo no queda limitado por la falta de memoria. Por ejemplo, considerando la figura 5.5 en la cual se muestra el tiempo de ejecución y el uso de memoria de FERMIN para el caso de estrés mostrando dos planificaciones distintas: paralela y secuencial. El uso de memoria para la planificación paralela es notablemente mayor, debido a que al principio del experimento varios procesos están siendo ejecutados en paralelo, y cada uno reserva una cantidad significativa de memoria. Por el contrario, la planificación secuencial muestra un menor uso de memoria (con picos de 40 GB vs. los 100 GB de la planificación paralela), pero a costa de casi doblar el tiempo de ejecución.

Por tanto, si los recursos de memoria son limitados, la paralelización de las tareas puede peligrar, debido a que será la falta de memoria y no el número de núcleos disponibles lo que va a limitar este paralelismo.

Para concluir, aunque la desagregación de recursos presenta unas penalizaciones claras cuando el ratio de operaciones no locales se incrementa, es decir, el acceso a recursos remotos y las latencias inducidas durante el proceso, sus puntos fuertes en términos de escalabilidad y distribución de recursos suponen un compromiso asumible y viable. De cara al futuro, es necesario atender las interesantes mejoras en los sistemas de interconexión, los cuales añadirían elasticidad a la unidad básica de construcción de los centros de datos tradicionales. Por tanto, se espera que el trabajo expuesto en este capítulo así como las evaluaciones realizadas ofrezcan información útil para la fase de refinamiento de la arquitectura *dReDBox* y al conocimiento en general en el área de investigación, aportando también las necesidades de los casos de uso de herramientas de análisis masivo de datos.

El trabajo expuesto en este capítulo será presentado en la conferencia *IEEE International Conference on High Performance Computing and Communications* durante los días 18 a 22 de Diciembre de 2017 en Bangkok, Tailandia, en representación del proyecto europeo *Disaggregated Recursive Datacentre-in-a-Box* (*dReDBox*).



*“ ’Twas twilight, and the sunless day went down;*
*over the waste of waters.”*
Ilustración de Thomas Seccombe. 1881

CAPÍTULO 6

# MONITORIZACIÓN DE REDES

## Metodologías simples para la detección de anomalías

Hoy en día los grandes centros de datos están formados por una serie de servicios y aplicaciones heterogéneas las cuales acostumbran a depender entre sí. Un ejemplo clásico es el de aquellos servicios web en los cuales el usuario realiza una petición contra el servidor mediante el navegador web, lo cual genera a su vez una petición del servidor web a las correspondientes aplicaciones asociadas, que del mismo modo consultan las bases de datos con el fin de obtener la información solicitada. Esta información realiza entonces el camino inverso transformándose durante el proceso hasta llegar al usuario.

En el camino han tenido lugar una serie de procesos subyacentes, como pueden ser, la comprobación de intrusiones, el paso a través del corta fuegos del sistema, o el análisis del sistema anti virus del centro de datos. Todo ello ocurre a través de la red, la cual dirige los mensajes de un componente a otro haciéndolos pasar entre distintos conmutadores, enrutadores, etcétera hasta llegar a su destino.

Cuando se produce una desviación en la calidad de servicio de alguno de estos componentes, esta repercute de forma negativa en la calidad de experiencia del usuario final, ya sea en forma de retardos o, en los casos más graves, con cortes del servicio. No obstante, averiguar qué componente está produciendo tal retardo no resulta sencillo en redes heterogéneas como las descritas. Y además, será necesario discernir si el componente es el causante mismo del retardo o si se está viendo afectado a su vez por el mal funcionamiento de otro componente del que dependan sus tareas.

Por todo esto, resulta esencial implementar un sistema de monitorización lo más omnisciente posible, **capaz de ofrecer información reactiva** con la que generar alarmas o visualizar desviaciones del comportamiento habitual, y en la medida de lo posible, reportar detalles **microscópicos** sobre la incidencia. Estos sistemas también pueden ofrecer otras **funcionalidades proactivas** como el análisis de tendencias, o la detección de problemas **macroscópicos** de largo plazo, como los expuestos en el capítulo 4º, *FERMIN*: Automatizando el Análisis del Tráfico.

Llegar a la causa raíz puede resultar muy complejo para un proceso automático de monitorización en tiempo real, pero se puede alcanzar un compromiso en el cual se ofrezcan pistas al analista o administrador de redes y servicios con las cuales acotar la zona afectada. En muchos casos la causa raíz será encontrada mediante la inspección manual de registros, o configuraciones en un conjunto de componentes candidatos obtenidos mediante los sistemas de monitorización.

Un sistema capaz de acotar una subred, máquina o VLAN concreta supondrá de por sí el ahorro de una gran cantidad de tiempo crucial en la resolución de la incidencia, reduciendo el tiempo de recuperación del sistema a su comportamiento habitual.

En este capítulo se exponen una serie de metodologías simples utilizadas durante el despliegue de un sistema de monitorización en la *Sociedad Estatal de Correos y Telégrafos* de España, como parte de un proyecto de *Naudit* comenzado en 2015, y que a día de hoy sigue evolucionando y aportando información esencial para la resolución de incidencias.

## 6.1. Introducción

Como se ha expuesto en la introducción y motivación de esta tesis, la supervisión de una red en grandes centros de datos está conlleva una serie de etapas que abarcan desde la recolección de datos de red o mensajes de *logs* de los distintos servicios presentes en la red, su transformación, almacenamiento, análisis, hasta, finalmente, la monitorización mediante sistemas de alarmas y visualización.

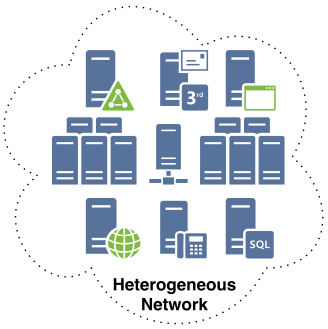



Figura 6.1
Red heterogénea.

A lo largo de esta sección se detallarán los distintos requisitos de un sistema de monitorización así como sus limitaciones y consideraciones a tener en cuenta durante el despliegue del sistema.

### 6.1.1. Proporcionalidad de la monitorización

Las redes de los grandes centros de datos están compuestas por numerosos y variados sistemas mediante los cuales se ofrecen distintos servicios. Estos sistemas dependen entre sí, y suelen estar agrupados formando distintas subredes y flujos de datos continuos entre distintas subredes. Todos estos servicios generan una cantidad de tráfico ingente que puede llegar a alcanzar varios TBytes diarios de datos, lo que supone un reto a la hora de monitorizar este tipo de redes.

Por tanto, la primera consideración a tener en cuenta durante la implantación de un sistema de monitorización, deberá ser la proporcionalidad de este, ya que, para la monitorización de un sistema no se puede requerir, haciendo una reducción absurdo, tantas máquinas como máquinas va a monitorizar. Deberá por tanto acotar y definir qué información es relevante y en qué puntos debe ser capturada.

Así mismo, las demandas de monitorización influirán directamente en el diseño del tamaño del sistema y su capacidad. Por ejemplo, la cantidad de datos a conservar para estudios proactivos de tendencias o líneas base repercute drásticamente en el tamaño de los sistemas de almacenamiento del sistema de monitorización. De la misma manera, el nivel de detalle y profundidad de los datos almacenados afectará al tamaño del sistema.

Por ejemplo, se puede decidir almacenar un día de tráfico completo en crudo para permitir su inspección microscópica en caso de incidencia, y a su vez conservar varios días de datos de los flujos de conexiones TCP del centro de datos. De la misma manera, conjuntos de estadísticas obtenidas a partir de estos mismos flujos o de otras fuentes (v. gr. *logs*, consultas SNMP), pueden ser almacenados durante varias semanas o meses con el fin de construir un registro histórico del estado de la red a partir del cual realizar tareas proactivas como la generación de líneas base o el análisis macroscópico de la red.

En el capítulo acerca del sistema *Loginson* se recogió la existencia de gran variedad de sistemas, y que además, desde el comienzo de esta tesis hasta el día de hoy han surgido muchas más que ofrecen multitud de funcionalidades y cada vez mejor rendimiento. No obstante, tanto en la experiencia académica como en la profesional a través de varios proyectos, se ha observado cómo se intenta hacer uso de determinadas tecnologías de forma burda, las cuales escalan principalmente de forma horizontal, malgastando recursos de forma innecesaria y muchas veces, sin comunicación entre departamentos, produciendo duplicidades en los sistemas e incluso incoherencias a través del uso de sistemas incompatibles entre sí. Estos sistemas de escalado horizontal, solo alcanzan el rendimiento buscado, cuando se hace uso de múltiples instancias en *cluster* [52], complicando aún más el despliegue del sistema.

No se debe despreciar tampoco el componente humano, y el problema de convencer de la utilidad del despliegue de determinados sistemas, sobre todo cuando estos sólo se usan ante la ocurrencia de problemas (v. gr. sistemas de copia de seguridad, sistemas de alarmas, detección de intrusiones, etc.) ya que no dejan de conllevar gasto de espacio y energía, lo cual no es algo despreciable. Como se visto en el capítulo anterior sobre arquitecturas desagregadas, el aprovisionamineto desproporcionado en los centros de datos repercute en gastos operacionales de la infraestructura.

Todo lo anterior pone de manifiesto que el sistema de monitorización a desplegar debe ser discreto y ocupar el menor número de recursos en el centro de datos a supervisar, siendo estos recursos tanto de espacio, energía, computación e incluso personal de la infraestructura. Esto conduce al siguiente apartado en el cual se discutirán los aspectos de intromisión del sistema de monitorización en los componentes monitorizados.

### 6.1.2. Sistemas pasivos versus sistemas activos

Junto con la proporcionalidad del sistema surge rápidamente el aspecto de la recolección de datos de los distintos componentes del centro de datos o red a monitorizar. A pesar de que existen sistemas *de facto* para la centralización y comunicación del estado de los sistemas, como por ejemplo *syslog* o SNMP, algunas alternativas se basan en la instalación de agentes en los servidores y componentes de la red, los cuales envían estadísticas a un punto central. Es decir, **sistemas de monitorización activos**. En primer lugar la instalación de estos agentes no siempre es posible debido a la gran variedad de sistemas operativos presente en las redes heterogéneas, lo cual supone un gran reto en términos de portabilidad.

El segundo obstáculo es, de nuevo, humano, ya que en empresas con sistemas externalizados suele haber distintas empresas responsables de cada departamento (v. gr. bases de datos, sistemas de correo, redes, seguridad, directorio activo, etc.) las cuales trabajan normalmente con restricciones muy altas de seguridad y confidencialidad. De forma concisa, no hay manera de garantizar si el agente puede suponer un riesgo para la calidad de servicio del componente monitorizado, o incluso, de que una vulnerabilidad ponga en jaque los sistemas mediante un uso desmesurado de los recursos de servidores de sistemas en producción.

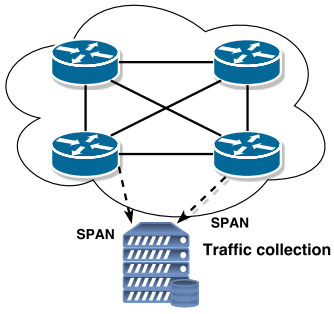



Figura 6.2
Simplificación del proceso de recolección de datos.

Este es de hecho, el tercer impedimento a considerar, los sistemas en producción no pueden permitirse compartir los recursos de computación y memoria con agentes de monitorización, que muchas veces suponen una doble monitorización debido a la presencia de sistemas propios del fabricante de cada sistema (v. gr. los sistemas propios de Cisco, Microsoft u Oracle de automonitorización).

Por otro lado, los **sistemas de monitorización pasivos** tienen la ventaja de no interferir en el funcionamiento normal del sistema, pero el inconveniente de que su omnisciencia es más limitada. En concreto, se puede realizar una replicación del tráfico de la red mediante un puerto espejo (*Switched Port Analyzer*, SPAN) que replica el tráfico recibido por el switch hacia otra interfaz (ver figura 6.2), pero esto solamente permitiría la monitorización del tráfico de forma pasiva, en la cual se tendría información sobre *logs* de aplicaciones (v. gr. registros de excepciones java, oracle, apache, etc.) o el estado interno de los componentes (v. gr. uso de CPU, memoria, espacio en disco, etc.).

Estos datos, en caso de necesitar recolectarlos, deberán obtenerse consultando su origen, allí donde han sido generados. Es preferible recolectarlos en momentos de baja utilización cuando los sistemas utilizan menos recursos, o de forma óptima, consultarlos bajo demanda durante los momentos en que se produzcan incidencias.

Una vez se ha establecido el método para la monitorización de las distintas capas, por ejemplo mediante consultas SNMP para la información de capacidad (v. gr. CPU, memoria, discos), recolección de tráfico, o copia de *logs*, será necesario acotar las zonas a monitorizar. Lo cual se trata a continuación.

### 6.1.3. Selección de los puntos a monitorizar

De forma intuitiva existen puntos críticos que deben ser supervisados desde el primer momento en que el sistema de monitorización está desplegado en la red. Algunos de estos sistemas son los referentes a las zonas del borde de la red, como los corta fuegos, sistemas de intrusiones, la zona desmilitarizada (DMZ) o perimetral en la cual se exponen los servidores que ofrecen servicios a la red externa, protegiendo la red interna de posibles intrusiones. Estos son puntos calientes en los cuales existe una alta incertidumbre ya que dependen del comportamiento de la red externa (v. gr. mayor número de visitas, tráfico no autorizado, etc.) son propensos a sufrir incidencias o comportamientos anómalos. Por otro lado, el núcleo de la red también resulta crítico para determinadas aplicaciones, como por ejemplo los sistemas de autenticación, bases de datos, etc. y debe ser supervisado para aportar información sobre aquellas incidencias que se produzcan desde la red interna empresarial (v. gr. cambios en bases de datos, aplicaciones, servicios, etc.).

Mediante los análisis proactivos de la red se pueden destacar zonas críticas donde haya retardos, retransmisiones o estadísticas con alta variabilidad que puedan afectar a la calidad de servicio. También, lógicamente, se añadirán aquellos sistemas que tras una incidencia deban ser monitorizados por no haberlo estado haciendo previamente, o aquellos sistemas demandados de manera explícita por el grupo de coordinadores.

No obstante, también debe acotarse de forma adecuada qué sistemas no monitorizar ya que monitorizar todo lo que ocurra en la red tiene un gran coste. La elección en base a subredes, VLANs, puertos, servidores, aliviará el proceso recolección, filtrando información innecesaria para conservar lo más relevante con la que obtener estadísticas para la monitorización y detección de anomalías. Actualmente, existe investigación en curso sobre el filtrado de tráfico a velocidades de 100 Gbps [86] y gracias a determinados controladores de tarjetas de red [31] se puede realizar capturas a 10 Gbps en múltiples interfaces. Esta velocidad es actualmente la velocidad más común en las interfaces de red empresariales.

## 6.2. Diseño del sistema de monitorización

En el caso del proyecto de sistema de monitorización desplegado en la *Sociedad Estatal de Correos y Telégrafos* de España en 2015 se realizó un despliegue de una sonda de captura convencional. A parte de esta sonda de captura se dispone también de una máquina virtual para el almacenaje de los datos procesados. En esta máquina virtual se realizará la visualización de los datos así como la creación de líneas base y generación de alarmas en base a los datos recibidos y el análisis proactivo realizado semanalmente.

Este despliegue sigue un esquema similar al desarrollado durante el proyecto *Loginson* en el cual mediante el controlador de tarjetas de red HPCAP [31] se realiza la captura de tráfico en varias interfaces a la vez, y con la ayuda otros sistemas y herramientas de *Naudit* se generan datos de flujos del tráfico en tiempo real. Toda esta información es almacenada en los sistemas de almacenamiento de alta velocidad, los cuales de forma programada (v. gr. por las noches) o en intervalos cortos (v. gr. cada 5 minutos) son procesados para extraer información de los datos recolectados. Esta información se transforma mediante programas ad-hoc y se transmite posteriormente a la máquina virtual de monitorización con el fin de importarla en una base de datos de análisis.

Con el fin de mantener el mayor nivel de omnisciencia del sistema monitorizado se optó no sólo por la captura de tráfico de distintas subredes y VLANs sino también por la recolección de los mensajes de *logs* de diversos servicios y de cada una de las capas que

lo componen. Por ejemplo, información de la capa de presentación (servidores web), de aplicación (métricas de servicios WAS, excepciones java), o de la capa de datos (bases de datos, excepciones oracle), entre otros.

Adicionalmente se extrae también información explícita del tráfico de las máquinas de las cuales se recoge información de sus aplicaciones, con el fin de poder relacionar problemas de red con incidencias en las capas superiores y viceversa. La información que se trata de obtener del *interior* de los servidores es la relacionada con el rendimiento de los servicios que ejecutan y no con la información o datos de los recursos que ofrecen. Por ejemplo, datos de utilización de memoria, CPU, número de hilos utilizados en las piscinas de hilos de las aplicaciones, entre otros.

De esta manera, en la monitorización de la red se trata de aumentar la granularidad al máximo. A parte de monitorizar VLANs enteras de tráfico y generar sus correspondientes estadísticas asociadas, se filtra de forma específica el tráfico de determinados servicios como los mencionados anteriormente o aquellos correspondientes a sistemas críticos como los sistemas de autenticación (Radius, LDAP). No obstante, resulta primordial mantener acotada la proporcionalidad de la monitorización ya que un grano demasiado fino puede repercutir en la generación de una gran cantidad de información que después no va a ser consultada, y que dañará el rendimiento de otra información más relevante.

### 6.2.1. Sonda de captura

La sonda de captura desplegada consta de 64GB de memoria RAM, un procesador Intel®Xeon®CPU E5-2640 v2 @2.60GHz con 8 núcleos cada uno y 4 sistemas RAID 0 de 15 TB cada uno para almacenar, en 3 de ellos, los datos capturados en las distintas interfaces, dejando el último para tareas de análisis, procesado de datos, y otras tareas.

Parte del estudio realizado durante el proyecto *Loginson* sirvió para desarrollar el sistema de monitorización desplegado en la *Sociedad Estatal de Correos y Telégrafos*. Para ello, se hizo uso del controlador de captura HPCAP [31] para la captura de tráfico y generación de registros a nivel de flujo. La recolección de otro tipo de información como los mensajes de *logs* se realiza mediante copias remotas (v. gr. SCP) programadas a distintos intervalos. Mediante otros sistemas de *Naudit* como *DetectPro* [1] se generan registros de subredes, VLANs o puertos de los correspondientes departamentos o servicios asociados.

[1] `http://www.naudit.es/en/detectpro/`

### 6.2.2. Máquina virtual para alarmas y monitorización

Durante el primer despliegue realizado en 2015, la máquina virtual desplegada en los sistemas de virtualización (ESX) de correos hacía uso de unos recursos modestos. En concreto, disponía de 8 GB de memoria RAM y 100 GB de almacenamiento para datos junto con 4 núcleos de procesamiento. A lo largo de los años este sistema ha ido almacenando mayor número de datos debido a la ampliación del número de sistemas monitorizados. En consecuencia los recursos se han incrementado progresivamente hasta disponer de 12 GB de memoria RAM, 200 GB de disco duro y 8 núcleos de procesamiento.

Una vez procesados los datos en la sonda de captura, estos son transferidos a la máquina virtual de monitorización para su inmediata indexación en la base de datos Elasticsearch. Esta base de datos, como ya se explicó en el capítulo dedicado a *Loginson* proporciona numerosas funcionalidades para el cálculo de estadísticas a partir de los datos, pudiendo realizar distintas operaciones de agregación, filtrado, y cálculos de estadísticas como medianas, desviaciones típicas, etcétera. Su gran versatilidad y flexibilidad permiten una integración sencilla con otras herramientas haciendo uso de su API REST a través de HTTP, o mediante librerías externas. Estas herramientas abarcan desde la visualización de los datos con Kibana o Grafana, hasta el desarrollo de herramientas propias en Python o Java para la importación y análisis de los datos.

Las herramientas de generación de líneas base y alarmas se han implementado

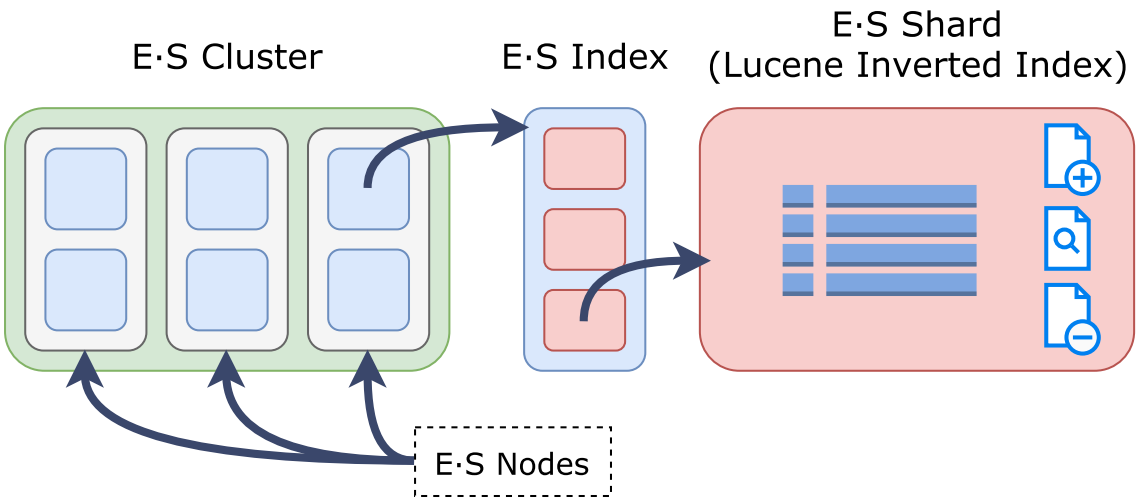



Figura 6.3: Diagrama del almacenamiento de datos en Elasticsearch.

mediante la librería de alto nivel *elasticsearch-dsl*[2,3] que facilita la programación de operaciones contra la base de datos Elasticsearch. No obstante, dado que Elasticsearch provee una API *RESTful* para la comunicación mediante HTTP y datos enviados en formato JSON, se podría realizar igualmente, aunque de forma más compleja, una comunicación con Elasticsearch sin necesidad de librerías externas.

De forma simplificada, la figura 6.3 muestra la distribución de la información en una base de datos Elasticsearch. Los elementos indexados en Elasticsearch se almacenan en un índice especificado a la hora de realizar esta operación. Estos índices tienen una estructura definida mediante un *mapping* en la cual se especifican los tipos de datos. Los índices se subdividen en un un grupo de *shards* que en esencia son índices invertidos de Lucene[4] en la cual, a *grosso modo*, se referencian palabras con los documentos que las contienen. A la hora de indexar un nuevo documento, Elasticsearch decidirá en qué *shard* almacenarlo de entre los que contiene su correspondiente índice.

Las distintas operaciones de aggregación, búsqueda, etc. se ejecutan a nivel de *shard*, y tener múltiples *shards* ofrece ventajas en términos de paralelismo (siempre hasta cierto punto). No obstante, el cometido esencial de los *shards* es su distribución entre un cluster de máquinas que estén ejecutando instancias de elasticsearch. Además de los *shards* principales, se pueden añadir *shards* replica con el fin de aumentar el rendimiento de búsqueda y la replicación de los datos en caso de fallo de uno de los nodos.

### 6.2.3. Visualización de los datos

Como se ha comentado previamente, para la visualización de los datos indexados en Elasticsearch se hace uso de Kibana, sistema mediante el cual se generarán gráficos de las distintas métricas monitorizadas. Kibana provee un entorno sencillo para la generación de centros de control (*dashboards*) mediante distintos gráficos de series temporales, histogramas, tartas, tablas, etc. Además, Kibana genera gráficos sencillos, lo cual facilita la visualización de los resultados, incidencias, desviaciones, etc. eliminando elementos redundantes o innecesarios [120] de los gráficos, evitando así que resulten densos o cargantes.

Existen otras herramientas de visualización alternativas, como *Grafana*[5] o Plotly[6] que permiten la generación de gráficos dinámicos para su visualización a través del

[2] `https://elasticsearch-dsl.readthedocs.io/en/latest/`
[3] `https://github.com/elastic/elasticsearch-dsl-py`
[4] `https://lucene.apache.org`
[5] `https://grafana.com/`
[6] `https://plot.ly/`

navegador web. Grafana ha tomado bastante impulso durante los últimos años, pero al comienzo del desarrollo del sistema desplegado en la *Sociedad Estatal de Correos y Telégrafos* disponía de muchas menos funcionalidades, aunque hoy en día supone un serio competidor para Kibana, ofreciendo mayor nivel de personalización de las gráficas.

Por otro lado, aunque *Plotly* se trate de un marco de trabajo (*framework*) más genérico, permite la generación de gráficas específicas, por ejemplo, para generar información adicional a aquella mostrada en los sistemas de visualización, y resulta un complemento ideal para los sistemas de alarmas, ya que genera ficheros HTML autocontenidos que pueden ser enviados entre departamentos. Su uso por tanto es distinto al de Kibana y Grafana, los cuales están pensados para trabajar con mayor cantidad de datos, agregando los datos automáticamente por ejemplo, y con mayor número de gráficos, junto con una capa de conexión con Elasticsearch.

No obstante, cada vez son más las alternativas disponibles para la exploración de datos de manera visual orientadas a inteligencia empresarial, monitorización de eventos y visualización de datos. Existen todo tipo de alternativas como *Dash*[7], *Tableau*[8], *Shiny*[9], *Motadata*[10], *Bokeh*[11] o *ggplot*[12] que permiten desde la construcción completa de la pila base de datos, controlador, visualización, hasta el desarrollo propio de soluciones visuales.

### 6.2.4. Magnitud actual de la monitorización

Desde finales del año 2015 al año 2016 la cantidad de información almacenada en la base de datos Elasticsearch diariamente pasó de los 100.000 documentos diarios al millón de datos diarios. Sin embargo, entre 2016 hasta la presente fecha, este número se ha incrementado hasta los 7 millones de datos diarios, que en tamaño en disco suponen unos 18GB por mes. La figura 6.4 muestra el crecimiento del número de datos indexados en los últimos 6 meses.

Debido al éxito del sistema de monitorización el proyecto ha continuado su expansión mediante la adición de una nueva sonda de tráfico, ya que la cantidad de tráfico monitorizado por la primera sonda superó entre las 3 interfaces los 2,5 millones de paquetes por segundo, 1,5 millones de conexiones y 10 Gbps. Actualmente esta segunda sonda monitoriza y procesa datos de otros muchas VLANs y servicios, los cuales son a

[7] `https://plot.ly/products/dash/`
[8] `https://www.tableau.com/`
[9] `https://www.rstudio.com/products/shiny/`
[10] `https://www.motadata.com/`
[11] `https://bokeh.pydata.org`
[12] `http://ggplot.yhathq.com/`

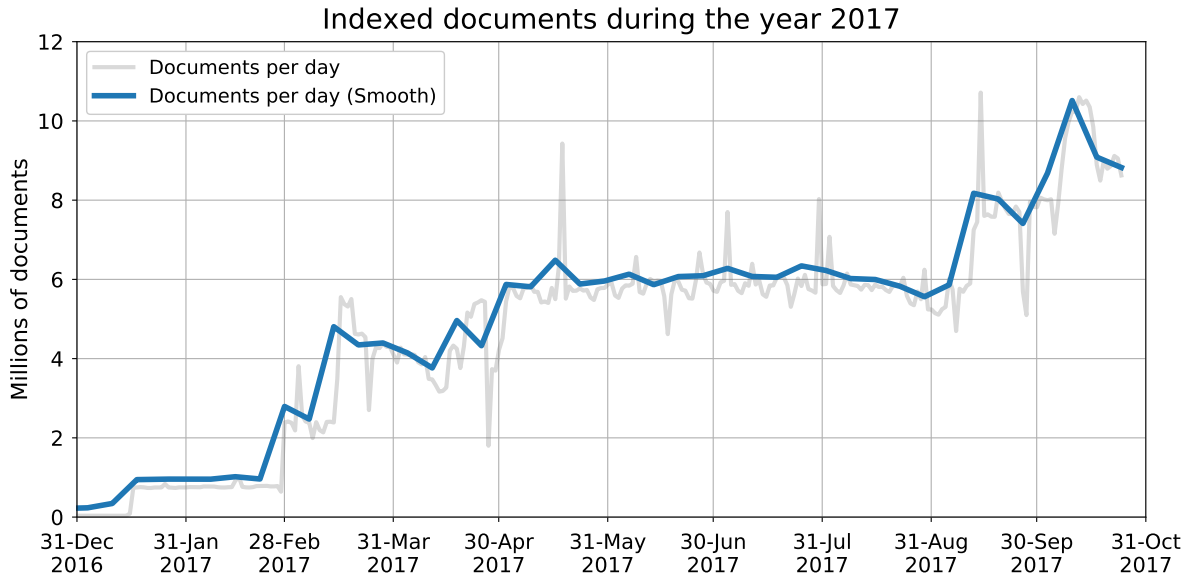



Figura 6.4: Número de documentos indexados diariamente en Elasticsearch.

su vez transmitidos a la máquina virtual de monitorización para su almacenamiento en la base de datos. Esta segunda sonda es más moderna constando de dos procesadores Intel®Xeon®CPU E5-2640 v4 @ 2.40GHz cada uno con 10 núcleos de procesamiento, 256 GB de memoria RAM junto con un sistema de almacenamiento de 4 RAIDs 0 cada uno con 30 TB de capacidad, uno por interfaz. A parte, un disco de análisis de 8 TB permite realizar tareas de procesamiento y análisis sobre los datos.

A día de hoy, entre las dos sondas se capturan más de 15 Gbps en las horas pico, con unos 2 millones de conexiones concurrentes y 3,5 millones de paquetes por segundo que permiten realizar una monitorización amplia de gran parte de la red de Correos. La recolección de nueva información continúa mediante la obtención de información del estado de los enrutadores y conmutadores mediante SNMP (uso de CPU, memoria utilizada, etc) con el fin de poder detectar tendencias a largo plazo en cuanto a la utilización y capacidad de los sistemas.

## 6.3. Metodologías para la detección de anomalías

Una vez desplegado el sistema de monitorización y recolección de información, comenzarán a actuar los sistemas reactivos y proactivos, como los sistemas de alarmas y de generación de líneas base, respectivamente. Generalmente, a pesar de realizar un filtrado inteligente de la información a capturar y de acotar los puntos supervisados, la

cantidad de información sigue siendo significativa y resulta esencial la utilización de sistemas de alarmas rápidos y eficaces.

Como se ha visto en capítulos anteriores, existen numerosas soluciones para la detección de anomalías y sus dependencias [71, 72, 111, 112], así como para la recolección de mensajes de *log* [52, 53, 55]. Algunas de estas alternativas son más o menos eficientes en el uso de recursos, unas utilizan sistemas pasivos y otras son más intrusivas. Pero por lo general, todos estos sistemas requieren muchos más recursos de los que sería recomendable utilizar, es decir, sobredimensionando el sistema de monitorización o realizando una recolección agresiva de los datos.

En cuanto a los sistemas de generación de líneas base y análisis estadístico existen soluciones complejas con gran trasfondo matemático [113–116] capaces de detectar sutiles desviaciones en el comportamiento de las métricas. No obstante, resulta indispensable disponer de un sistema de comprobación capaz de comprobar los numerosos sistemas monitorizados y sus correspondientes series temporales de la manera más rápida posible, informando puntualmente de las incidencias ocurridas.

Además, de nuevo, el componente humano llama a la generación contenida de alarmas, ya que la reiteración de alertas, aún justificadas, produce desidia por parte de los administradores de sistemas, silenciando futuras advertencias y perdiendo confianza en el sistema de alarmas. Es decir, dado que lo más grave del escándalo es precisamente acostumbrarse a el, resulta preferible generar únicamente alarmas de desviaciones graves a costa de no informar de otras desviaciones menores, salvo que estas se repitan en el tiempo o bien, controlar adecuadamente los intervalos entre la primera alerta y las sucesivas notificaciones.

El interés en esta clase de sistemas de detección de anomalías se ha incrementado en los últimos años con el crecimiento de los centros de datos y los servicios alojados en la nube. Sin embargo muchos de estos sistemas requieren o bien un sistema de aprendizaje automático complejo, o la instalación de agentes en las máquinas a supervisar. Algunos sistemas desarrollados posteriormente al método expuesto en este capítulo se describen a continuación.

Por ejemplo, el sistema propuesto por Zhang et al. [117] trata de detectar la causa raíz de las incidencias y las anomalías mediante el análisis del comportamiento de los hilos de las aplicaciones ejecutadas en los sistemas y el uso que hacen de los recursos. La obtención de esta información requiere la instalación de agentes Python en las distintas máquinas, lo cual no es una tarea sencilla cuando los distintos departamentos de un gran centro de datos son gestionados por compañías distintas. Debe tenerse en cuenta también

que la instalación de un agente en las máquinas no suele estar permitido, ya que puede suponer vulnerabilidades inesperadas o incrementar el uso de los recursos. En el sistema propuesto tampoco se explica la manera en que se centraliza toda la información, lo cual tampoco resulta trivial debido a los distintos componentes de seguridad diseminados por los centros de datos (v. gr. corta fuegos, sistemas de intrusiones, enrutadores, etc.).

Muchas aplicaciones ya proveen registros sobre el uso de los recursos (v. gr. CPU, memoria, hilos) con gran nivel de detalle. Generalmente la dificultad reside en la recolección de toda esta clase de registros para realizar una monitorización posterior o una correlación de eventos que pueda indicar la localización de la causa raíz. De no ser considerados, estos registros quedan en la máquina a la espera de que el administrador de sistemas los consulte durante la búsqueda manual de causas.

Por otro lado, sistemas como el propuesto por Calheiros et al. [118] hacen uso de la técnica *Isolation Forest* [119] propuesta por Liu et al. para la búsqueda de anomalías. El método *Isolation Forest*, explora el hecho de que las anomalías tienden a ser más escasas y estar aisladas de las observaciones normales. Para ello elige aleatoriamente las características a tener en cuenta y las particiona aleatoriamente de forma recursiva en una serie de particiones.

El método aprovecha estas características de las anomalías para aislar los valores anómalos cerca de la raíz de árboles binarios, usados para representar los datos disponibles. No obstante, en sus propias palabras, este método no resultó satisfactorio en sus primeros resultados (*"Initial attempts of applying iForest with such small number of attributes did not lead to good anomaly detection power on the approach."*). Por ello, derivaron un nuevo grupo de atributos a partir de los datos observados previamente, generando unos atributos con los *valores esperados* o el *comportamiento habitual*. Para ello hace uso de un método muy similar al que se explicará en las siguientes secciones. Aprovecha las cualidades cíclicas y estacionales de los datos agrupándolos en ciclos semanales y subciclos de una hora, guardando la mediana de los datos de cada subciclo, conformando de esta manera una nueva serie de datos con los valores *esperados* de la serie temporal.

### 6.3.1. Premisas sobre la creación de líneas base

Ateniéndo a lo expuesto anteriormente, deberán tenerse en cuenta otros detalles referentes al tipo de datos a monitorizar. Por ejemplo, no se comportan de la misma manera métricas como el volumen de tráfico de una determinada subred, el número de excepciones de sistemas Java, o el uso de CPU o memoria en los componentes monitorizados. No

será necesario, por ejemplo, crear una línea base que defina el comportamiento habitual del uso de CPU y memoria si lo que se pretende es alertar cuando el uso se acerque al máximo soportado por el componente. Si por el contrario se quisiese alertar de un cambio en la tendencia habitual del uso de memoria, por ejemplo, que un componente lleve una semana usando el doble de memoria de lo habitual durante varios días seguidos. En este ultimo caso se podría alertar de un cambio en la utilización de memoria, requiriendo acciones sobre la capacidad de los sistemas utilizados. Por otro lado, el primer caso sólo necesitará comparar el valor actual con el máximo soportado, alertando inmediatamente en caso de llegar, por ejemplo, al 90 % de utilización.

El sistema de líneas base desplegado en la *Sociedad Estatal de Correos y Telégrafos* tiene dos propósitos: En primer lugar, la visualización gráfica de desviaciones respecto al comportamiento habitual. Este caso es de gran utilidad para aportar evidencias de una posible incidencia, algo necesario para que los departamentos encargados de los sistemas monitorizados actúen u observen las consecuencias de realizar determinados cambios. Por ejemplo, si de forma habitual los tiempos eran altos, podrán demostrar cómo un cambio ha repercutido en la mejora de los tiempos de respuesta. Por otro lado, el segundo caso sirve para la generación de alarmas que permitan realizar acciones correctivas lo más rápido posible.

En esta sección nos centraremos en la generación de líneas base mediante metodologías simples para el caso de la monitorización del tráfico de red. Un aspecto esencial

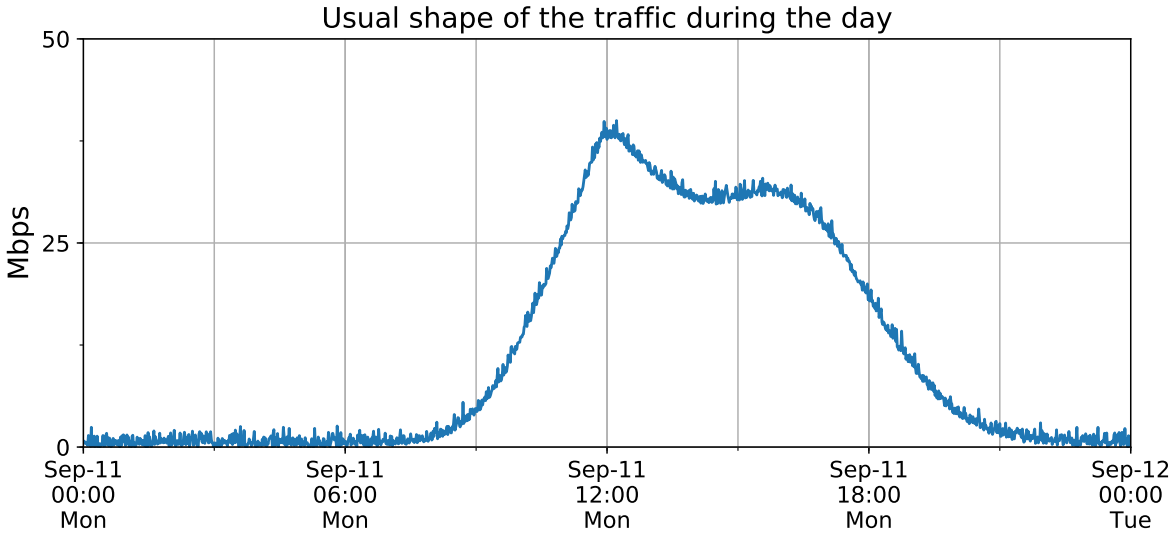



Figura 6.5: Forma del tráfico durante un día habitual.

en este proceso es el comportamiento del tráfico durante el día, atendiendo a las horas valle y horas puntas del sistema. En la figura 6.5 puede observarse la silueta del tráfico durante un día habitual laboral, estando el pico de tráfico diario en torno al medio día.

Por supuesto, este comportamiento varía no sólo durante el día y la noche sino también en función de los días de la semana, siendo distinto durante los días laborales que durante los fines de semana. Anualmente también se pueden apreciar variaciones en los días festivos o durante las épocas de verano o Navidad. Sin embargo, en las líneas base desarrolladas se tendrán en cuenta únicamente los cambios de comportamiento de los días laborales y fines de semana así como los diurnos y nocturnos.

### 6.3.2. Proceso de creación de la línea base

Cuando el histórico de datos disponibles es suficientemente amplio se podrá comenzar a realizar análisis proactivos para, en este caso, la generación de líneas base que definan el comportamiento habitual de las series temporales analizadas. Para ello, y con el fin de realizar este proceso de la manera más rápida y efectiva, se tomarán en cuenta un cierto número de semanas enteras de datos. Cuanto mayor sea el número de semanas a tomar en cuenta, más rígida será la línea base generada, resultando más difícil de consolidar un cambio en la silueta de la línea base.

Por ejemplo, si tras el despliegue de un nuevo servicio en la red monitorizada hubiese un cambio en la cantidad de tráfico, una línea base definida mediante los datos de las últimas ocho semanas de datos tardaría en adaptarse a este nuevo comportamiento habitual. Por otro lado, una línea base que utilizase sólo unas pocas semanas de datos podría viciarse rápidamente y añadir desviaciones reiteradas del tráfico a su silueta. Por ello, el sistema desarrollado permite parametrizar el número de semanas a utilizar con el fin de variarlo dependiendo de la necesidad a cubrir.

En esta sección tomaremos como ejemplo la figura 6.6 para explicar el proceso de generación de las líneas base. Se tomarán cuatro semanas de datos previos a la semana actual donde el comportamiento, es bastante estable. Como puede apreciarse, durante el fin de semana el tráfico es menor, e incluso el viernes es más bajo que durante el resto de días laborales. En particular, la resolución de estos datos será de dato por minuto, lo cual no descarta que algunos sistemas sí se supervisen a resolución de segundo. Esta resolución resulta adecuada en términos de almacenamiento y rendimiento ya que se entiende que en el intervalo de un minuto no ocurre nada crucial para la supervisión de los sistemas que justifique monitorizar el sistema a resolución de segundo. De hecho, de ser así siempre se podría importar el valor máximo en el intervalo de un minuto, ya

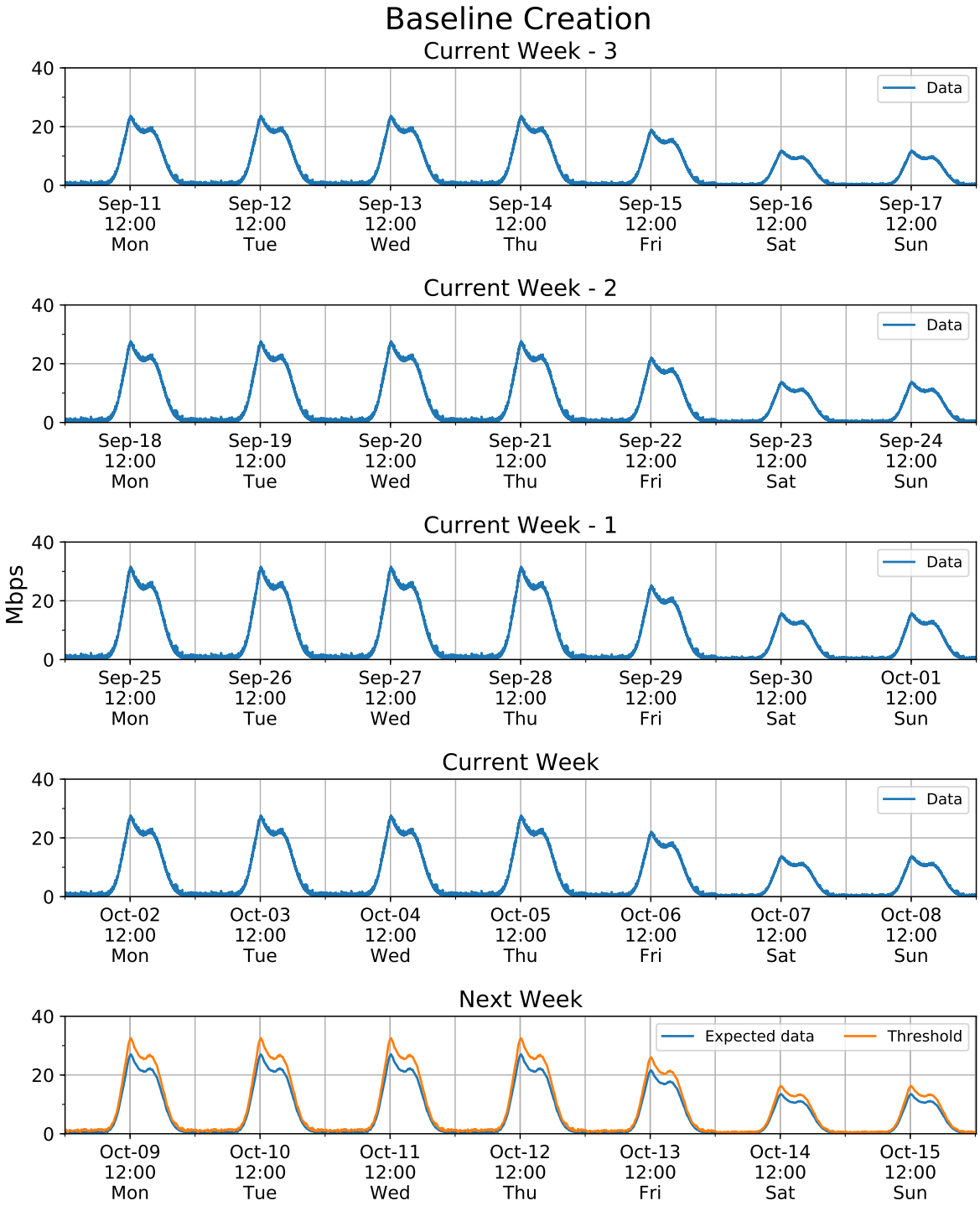



Figura 6.6: Creación de una línea base.

que la generación de alarmas en tiempo real absoluto resultaría extraordinariamente compleja, mientras que la notificación de incidencias en intervalos de unos pocos minutos resulta mucho más fácil de abordar.

Para el cálculo de la línea base se agruparán los datos pertenecientes al mismo día de la semana y a la misma hora y decena de minutos. Esto repercutirá en la resolución real de la línea base. Si la serie temporal original tiene un dato por minuto, la línea base tendrá un dato por cada diez minutos, aunque obviamente, se puede interpolar mediante distintos métodos para poder tener una serie con la misma resolución que la original. Intuitivamente, este agrupamiento se realiza debido a que todas las 8 de la mañana de los lunes son iguales, de tal forma que el comportamiento habitual del próximo lunes a las 8 de la mañana deberá muy similar al comportamiento de los anteriores lunes a las 8 de la mañana. El agrupamiento en grupos de 10 minutos permite tener mayor variabilidad en los datos con el fin de generar un dato de mayor calidad.

Por tanto, para generar el dato de las 8 de la mañana de la línea base mediante 4 semanas de datos a resolución de minuto, dispondremos de 40 datos. Con el fin de proveer un valor robusto se calculará el dato mediano. La mediana tiene un punto de quiebre del 50 % lo que evita que se desvirtúe fácilmente ante la presencia de unos pocos valores atípicos (*outliers*). El punto de quiebre es la proporción de valores atípicos u observaciones incorrectas que un estimador, como la mediana o el promedio, puede manejar antes de dar un resultado incorrecto. En el caso de la media, su punto de quiebre es del 0 % ya que un único valor atípico puede alterar considerablemente el valor resultante. El mismo proceso se seguirá para el siguiente dato, el de las 8:10 de la mañana, y así sucesivamente hasta tener generar la serie temporal completa de la línea base.

Además de calcular una serie temporal con el comportamiento *"esperado"* estimado mediante la mediana de los datos anteriores, se calculará también una serie con la desviación típica de los valores con el fin de poder definir un umbral que ayude a la generación de alertas, eliminando posibles falsos positivos. Un estimador robusto de la desviación típica sería la desviación absoluta respecto a la mediana (MAD en sus siglas en inglés, *Median Absolute Deviation*). No obstante, dado que la obtención de los datos se hace mediante las funciones provistas por Elasticsearch (mediana, promedio, desviación típica, varianza, etc.) no se ha hecho uso de este estimador, aunque no se descarta su uso en el futuro, e incluso se ha propuesto [13] como característica al equipo de Elasticsearch.

De hecho, durante el desarrollo del sistema de líneas bases se ha debido verificar apropiadamente el cálculo realizado de Elasticsearch de las distintas estadísticas, ya que

[13] `https://github.com/elastic/elasticsearch/issues/26681`

en algunas ocasiones el resultado provisto en versiones anteriores de la base de datos no ha sido el que cabría esperar, obteniendo resultados incorrectos. Estos problemas han sido notificados [14] al equipo de desarrollo de Elasticsearch, quienes han corregido estos problemas en versiones posteriores de Elasticsearch.

Dada la naturaleza de Elasticsearch, los documentos almacenados en un mismo índice pueden corresponder a múltiples series temporales o a una sola, en función de la agregación que deseemos. Por ejemplo, si en los datos del índice *correos_logs* almacenamos información y diversas métricas correspondientes a varios servicios, capas, etc., deberemos filtrar toda esta información almacenada en el índice con el fin de obtener la serie temporal de la capa, servicio y métrica que nos interesa, o por el contrario estaríamos obteniendo una serie temporal agregada de varias capas, servicios, etc.

Por ello, a la hora de generar una línea base para una serie de métricas de un servicio en concreto, deberán especificarse todas las tuplas clave, valor, que definan la serie temporal deseada. Por ejemplo, la nota de código 6.1 muestra un ejemplo de fichero de configuración para la generación de líneas base. En primer lugar, una serie de opciones definen el intervalo de datos a seleccionar, que abarca desde el momento especificado en "gt" hasta "lt" ("mayor que", y "menor que", en inglés, *"greater than"* y *"lower than"*, respectivamente), utilizando el parámetro *"date_field"* para indicar el campo correspondiente a la fecha.

Después, se provee una lista de métricas de las cuales se van a generar líneas base, así como una serie de tuplas clave valor que sirven para filtrar la información, y, en el caso del ejemplo, considerar únicamente la información del servicio web y la capa de presentación. Esta misma información deberá añadirse de nuevo como datos extra para poder visualizar correctamente la línea base generada junto con los datos originales. Dos parámetros *"data_index"* y *"baseline_index"* definen el índice del cual se van a extraer datos y el índice donde van a introducirse los datos generados. Como puede observarse, en este caso, los índices *correos_logs* y *correos_baseline_logs* podrían ser consultados simultáneamente mediante el patrón de índices *correos*logs*, lo que permitiría acceder a la información de la línea base y de los datos reales en una misma consulta, facilitando las tareas de visualización y generación de alarmas.

[14] `https://github.com/elastic/elasticsearch/issues/17362`

```
{
  "gt"           : "now-4w",
  "lt"           : "now",
  "date_field"       : "fecha",
  "data_index"   : "correos_logs",
  "data_interval" : "10m",
  "data_op" : "median",
  "fields" : ["tiempo de respuesta", "peticiones recibidas"],
  "query"    : {
    "servicio"   : "web",
    "capa"       : "presentacion"
  },
  "extra_data": {
    "servicio"   : ("web", "string"),
    "capa"       : ("presentacion", "string")
  },
  "baseline_index" : "correos_baseline_logs"
}
```

Nota de código 6.1: Ejemplo de fichero de configuración para una línea base.

### 6.3.3. Generación de alarmas

Una vez construida la línea base, se puede proceder a la utilización de estos datos para la generación de alarmas. Para ello se ha implementado una librería Python mediante el API *elasticsearch-dsl* comentada anteriormente. Esta librería permite crear objetos *GenericAlarm* (Alarma Genérica) con una serie de atributos que definen el índice a consultar (o patrón que abarca varios índices), los parámetros con los cuales solicitar información filtrada a Elasticsearch así como las condiciones para generar una alarma. En concreto, las condiciones para generar una alarma pueden estar basadas en el número de veces consecutivas que se cumple una condición; la ventana temporal en la cual se satisfacen, y el tiempo mínimo a mantener entre alarmas.

Estas alarmas genéricas permiten establecer un umbral numérico (v. gr. 90 % de uso, para el caso de métricas de memoria), o bien un umbral basado en una función que haga uso de uno o varios campos presentes en los documentos. Este segundo caso es el que se

emplea para la generación de alarmas mediante líneas base ya que se puede establecer una condición como la que se observa en la nota de código 6.2, definida mediante una función lambda, haciendo uso de múltiples métricas.

Las alarmas desarrolladas permiten realizar operaciones con gran versatilidad en tres facetas distintas. Por un lado, temporal, ya que permite la selección de intervalos concretos de tiempo, y la elección de la agregación temporal (muestreo) de los datos seleccionados, así como establecer condiciones temporales como ventanas temporales para la ocurrencia de incidencias o intervalos de notificación. Del mismo modo, ofrece flexibilidad en cuanto al filtrado de los datos, permitiendo establecer alarmas de grano fino o grueso. Por último, permite definir funciones personalizadas que hagan uso de las métricas presentes en los documentos indexados para establecer condiciones de alarma.

La figura 6.7 muestra dos ejemplos de desviaciones sobre el comportamiento normal. La primera, destacada mediante un círculo rojo, ocurre antes del medio día, y muestra cómo en el intervalo de una cierta ventana temporal (en amarillo) ha ocurrido una desviación que supera no solo el valor esperado sino también el umbral definido mediante la desviación estándar. En este caso, se deberá generar una alarma notificando del comportamiento anómalo de la métrica monitorizada. Por el contrario, la segunda desviación, ocurrida después del medio día no llega a superar este umbral, y por tanto no requiere notificación alguna, ya que se mantiene acotada dentro de los márgenes de la variación habitual de los datos.

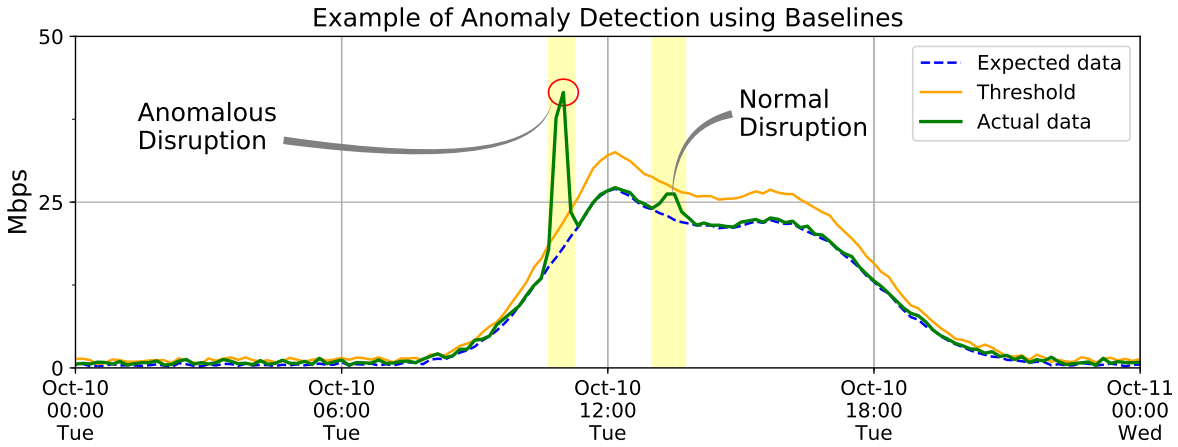



Figura 6.7: Detección de anomalías mediante líneas base.

```
from elastic_alarms.generic_alarm import *

client = Elasticsearch("localhost", timeout=240) #elastic connection
client.indices.refresh(index="data*network") #refresh database

gte = "now-30m" #data interval
lte = "now"
main_field = "ongoing_flows"
reference_field  = "incoming_flows"
timefield  = "fecha"
index = "data*network"

vlans = ["vlan1", "vlan2", "vlan3", "vlan4"]

f = lambda d: (d["ongoing_flows"] == 0) & (d["incoming_flows"] > 10)

for vlan in vlans:
    ext_data = {
      "vlan" : (vlan, "string"),
      "layer" : ("vlans", "string"),
      "group": ("radius", "string"),
      "service": ("network", "string")
    }

    query = Q("match", vlan = vlan)
    id_ = "{}_{}_critical".format(vlan, reference_field)

    fca = FieldComparisonAlarm(id_, client, main_field, \
            [reference_field] , index, timefield, \
    queries=[query], raise_if=f)

    fca.setComparisonMessage("eq")
    fca.setDataSamplingInterval(1)  #minutes
    fca.setAlarmSamplingInterval(5) #
    fca.setMinConsecutiveMatches(5) #number of consecutive alarms
    fca.setExtraData(ext_data)
    fca.setSearcher(gte, lte)
    fca.printUniqueAlarms("data-network-alarms", "alarms")
```

Nota de código 6.2: Ejemplo de definición de alarma en Python.

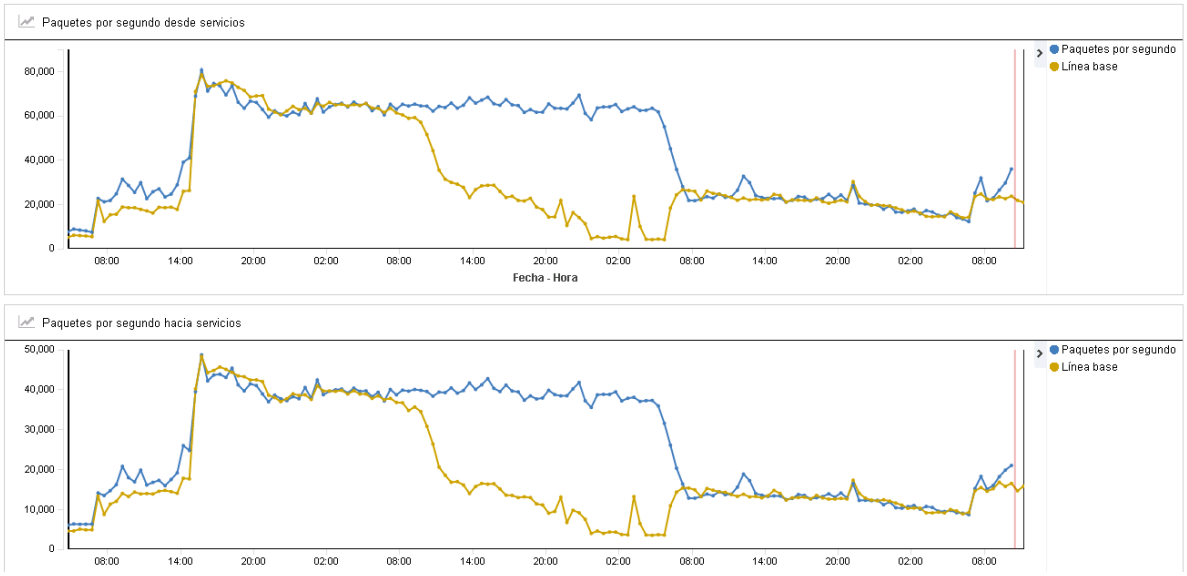



Figura 6.8: Evolución de los paquetes transmitidos en una determinada VLAN.

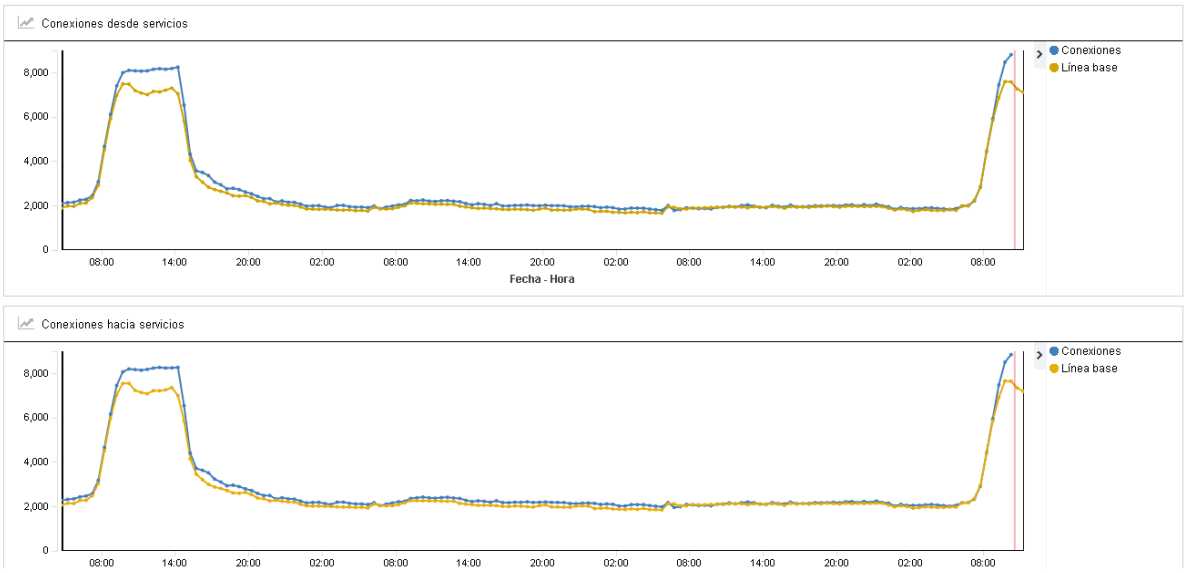



Figura 6.9: Estado de las conexiones entrantes y salientes en una determinada VLAN.

### 6.3.4. Ejemplo práctico I: Copia de seguridad a deshora

A continuación se describe un primer ejemplo práctico del uso del sistema desarrollado, ocurrido en el entorno de despliegue mencionado al inicio del presente capítulo. Entre los días 9 y 10 de Septiembre el sistema de alertas notificó, mediante la comparación de los datos reales con el valor esperado definido por la línea base, de un incremento anómalo en el número de paquetes entrantes y salientes para una determinada VLAN. Este caso de uso, por tanto, emplea los datos obtenidos a partir del análisis del tráfico, su disección, etc.

Ante esta notificación, el analista o responsable acudiría en primera instancia a los sistemas de visualización con el fin de observar el comportamiento de manera visual y ver cómo han sido afectadas otras métricas del sistema monitorizado. La información ofrecida por las gráficas en este primer vistazo resulta esencial, ahorrando un tiempo crucial al analista en la resolución de incidencias así como en la valoración de la gravedad de estas, pudiendo priorizar su trabajo de una manera más eficiente.

En este caso, la figura 6.8 muestra el incremento producido en el número de paquetes transmitidos de forma entrante y saliente, entre las 8 de la mañana de un día y el siguiente. Sin embargo, en la figura 6.9 se observa que el número de conexiones establecidas, tanto entrantes como salientes, se mantiene dentro del comportamiento habitual durante el intervalo afectado, y no se observa un aumento considerable del número de conexiones, aunque el valor parece situarse por encima de la línea base.

Con esta información, un analista podría concluir que el incremento en cuestión no parece estar producido por una gran cantidad de conexiones, sino por unas pocas en particular que generan gran cantidad de tráfico. En este punto, disponer de información adicional puede ahorrar aún más tiempo a la hora de averiguar qué servicio está produciendo el incremento de paquetes transmitidos. En este caso se dispone también de información sobre el puerto contra el que se realizan las conexiones. Mediante la selección del intervalo temporal de tiempo relevante a la incidencia, el resto de figuras recalculan sus valores, ofreciendo nueva información.

Como se observa en la figura 6.10 el puerto 8400 destaca significativamente en cuanto a la cantidad de bytes salientes transmitidos. Este puerto corresponde a un servicio de copias de seguridad (*backup*), lo cual explica la subida repentina de tráfico. Posteriormente se notificaría de lo ocurrido, y los responsables tomarían las medidas oportunas en caso de que el comportamiento del sistema de copias de seguridad no haya sido el apropiado. Habiendo ocurrido durante el fin de semana (obsérvese el patrón de conexiones establecidas), es posible que fuese una copia de seguridad puntual.

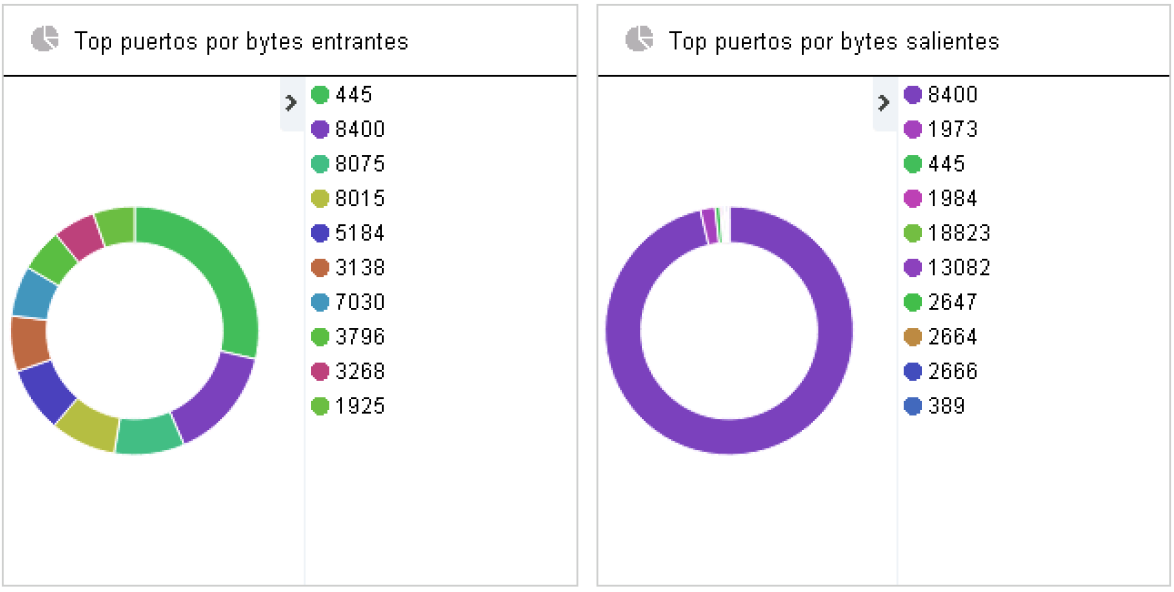



Figura 6.10: Puertos con más bytes transmitidos en una determinada VLAN.

### 6.3.5. Ejemplo práctico II: Geolocalización de las IPs visitantes

En este segundo caso de uso se propone la realización de un mapa de visitas a partir de la geolocalización de las direcciones IP. De esta forma se podrá visualizar si se reciben visitas de países que no estén contemplados entre los clientes objetivo del servicio ofrecido por la compañía. Entre los retos de este caso de uso está el ser capaz de geolocalizar las direcciones IP sin hacer uso de servicios externos que requieran conexión a internet por parte de los sistemas de monitorización.

A diferencia del primer caso de uso, en esta ocasión se empleará la información obtenida a partir del análisis de los mensajes de log de los servidores Apache y la disección de trazas mediante *HTTPanalyzer* (descrito en el capítulo 2º), los cuales han sido posteriormente indexados en Elasticsearch para la visualización de distintas estadísticas, como por ejemplo la evolución del número de visistas de una determinada IP en el tiempo. En concreto, se almacena el top 10 IPs para cada minuto, junto con el número de consultas realizadas para cada servicio, de entre un grupo de servicios web.

Da manera complementaria, se pueden realizar otro tipo de visualizaciones *ad-hoc* de gran valor para los departamentos implicados en la seguridad y los servicios web, que son más complicadas de generar mediante Kibana. En este caso, Kibana requiere conexión a internet para mostrar el mapa sobre el que pintar los datos, y en el entorno analista donde se ha desplegado el sistema de monitorización, el tráfico exterior está

altamente controlado. Así mismo, la obtención de datos de geolocalización requiere una funcionalidad adicional que ni Elasticsearch ni Kibana proporcionan, y que deberá ser suplida por otras vías.

Para solventar estos obstáculos se hará uso de distintos métodos. En concreto, la librería Plotly permite generar mapas (Choropleth Maps[15]) mediante Python o JavaScript. Y por otro lado, la información para geolocalizar direcciones IP a nivel de país proporcionada por *ip2location.com*[16]. Este conjunto de datos proporciona el nombre de país y código de país (según el *ISO 3166*[17] alpha-2) en rangos de IPs. Por otro lado, se empleará también una lista para traducir códigos de países del ISO 3166 alpha-2 al ISO 3166 alpha-3. Este ultimo formato es el soportado por Plotly para la identificación de regiones. La aplicación web desarrollada permite seleccionar un intervalo de tiempo concreto y generar el mapa a petición por el usuario. Así mismo, se ha desarrollado también en Python como herramienta más avanzada para los analistas.

La generación de estos mapas puede estar condicionada a alertas que indiquen que varias IPs están realizando un gran número de peticiones en un corto intervalo de tiempo. De esta forma se podría distinguir claramente si el origen de esas visitas es *a priori* legítimo (siempre en relación al servicio prestado) o anómalo. Existen otros conjuntos

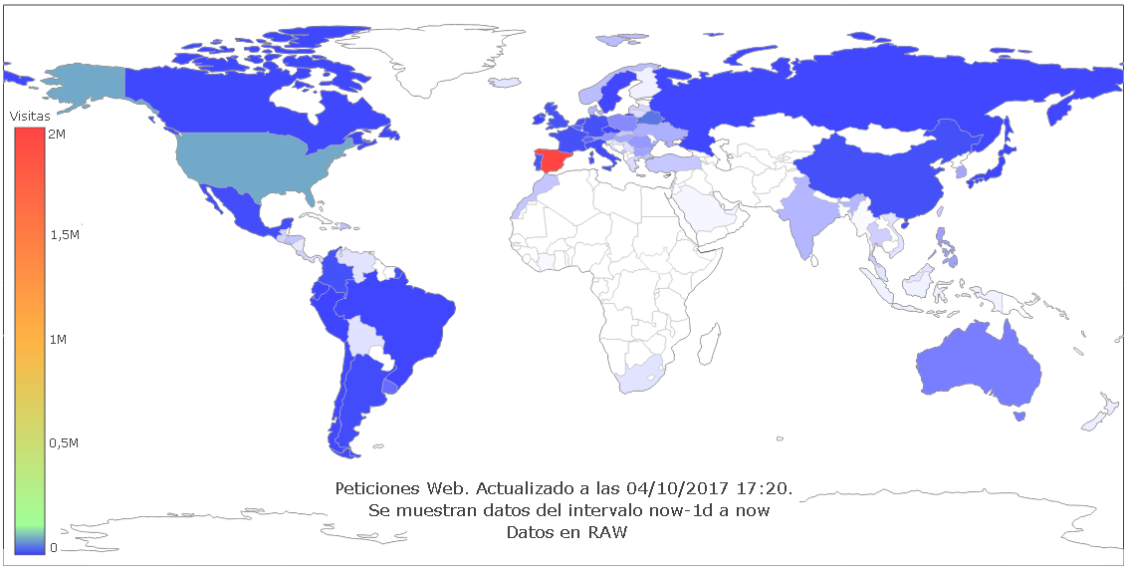



Figura 6.11: Visualización de las visitas web mediante la geolocalización de IPs.

[15] https://plot.ly/python/choropleth-maps/
[16] http://lite.ip2location.com/database/ip-country
[17] https://www.iso.org/iso-3166-country-codes.html

de datos con mayor información para acotar a nivel de región, ciudad o código postal el origen de los datos. También existen otros conjuntos de datos[18] con información sobre Proxys conocidos de VPNs, DCHs o salidas TOR que pueden ser empleados para evitar el tráfico de determinados sistemas. Con el fin de evitar aquellas visitas provenientes de subredes internas de la compañía, se dispone de una serie de filtros a nivel de subred para eliminar esas IPs del conjunto de datos analizado.

La visualización desarrollada genera un mapa mundial que considera hasta 100,000 IPs distintas, obteniendo aquellas con más visitas en el intervalo consultado. Un mensaje emergente muestra las 10 IPs con más peticiones de cada país. La figura 6.11 muestra un ejemplo de los mapas interactivos descritos anteriormente. El rendimiento de esta solución permite generar el mapa interactivo con los datos de visitas de la ultima semana en menos de 1 minuto para un determinado servicio. La cantidad de datos considerados por elasticsearch para todos los servicios en conjunto (de un total de 4), es aproximadamente de 400,000 datos semanales. Por tanto, en el peor de los casos, podría haber hasta 100,800 IPs distintas en una semana para un único servicio. Adicionalmente se produce un fichero CSV con las IPs consideradas durante el análisis junto con su información de geolocalización y número de visitas.

De manera similar a los otros casos de uso expuestos durante este capítulo, se podría generar una serie de líneas base o datos de referencia para cada país de manera que en caso de producirse un incremento súbito de visitas provenientes de un determinado país, se alertase a los equipos responsables. Este sistema hipotético requeriría la adición de la información de geolocalización durante la indexación de los datos para facilitar la generación de alarmas en tiempo real.

### 6.3.6. Rendimiento del sistema de alarmas

Actualmente, de forma semanal, el sistema de generación de líneas base calcula 598 series temporales a partir de las 8 semanas previas de datos, con datos a resolución de minuto, lo cual supone más de 48 millones de datos a considerar. Estas 598 líneas base están descritas en 126 ficheros como el que se muestra en la nota de código 6.1 y se corresponden con diversas métricas de distintos servicios monitorizados.

El tiempo de generación completo de las series temporales es, a día de hoy, de una hora y cuarto. La figura 6.12 muestra la estado de la carga de la máquina virtual durante el cálculo de las líneas base a distintas resoluciones.

[18] `http://lite.ip2location.com/database/px1-ip-country`

Así mismo, cada dos minutos el sistema de alarmas comprueba los datos más recientes, (entre 30 y 60 minutos anteriores), de un total de 303 series temporales en busca de anomalías. El tiempo para realizar esta comprobación no supera los 20 segundos, y aun en los momentos con más carga se mantiene acotado por debajo del minuto. Lo que supone, en el mejor de los casos, una velocidad de comprobación de 655 elementos por segundo. Todo este proceso se hace de forma paralela a la indexación en tiempo real de datos, que supone hasta 4,800 elementos por minuto, así como la consulta habitual de los datos por los equipos de analistas, que generan consultas sobre la base de datos. Como se vio en la sección 6.2.4, actualmente se indexan hasta 7 millones de datos diarios.

En el capítulo 6.3 se vieron algunos sistemas que hacen uso de un método similar de generación de atributos en base a los datos observados anteriormente. En concreto, el sistema propuesto por Calheiros et al. [118] hacía uso de datos agregados cada 30 minutos. El rendimiento observado por Calheiros et al. fue el siguiente: el cálculo de las puntuaciones de anomalía (*anomaly score*) para 2722 llevó tan solo 28 milisegundos usando 22.8MB de memoria RAM, con un tiempo de entrenamiento de 579 milisegundos (*"calculation of anomaly scores of 2722 data points [...] took 28 milliseconds [...] The training time was in 579 milliseconds..."*).

Lo que no se tiene en cuenta en el artículo mencionado, es el tiempo que conlleva consultar los datos alojados en una base de datos en continua consulta por otros sistemas. Tampoco tiene en cuenta el tiempo de cálculo de los atributos adicionales a partir de los datos previos (*"This has been solved with the use of another set of derived attributes that encodes the expected seasonal behavior of the metric (based on previous observed values)."*). Por último, no se considera el hecho de que habitualmente las métricas de

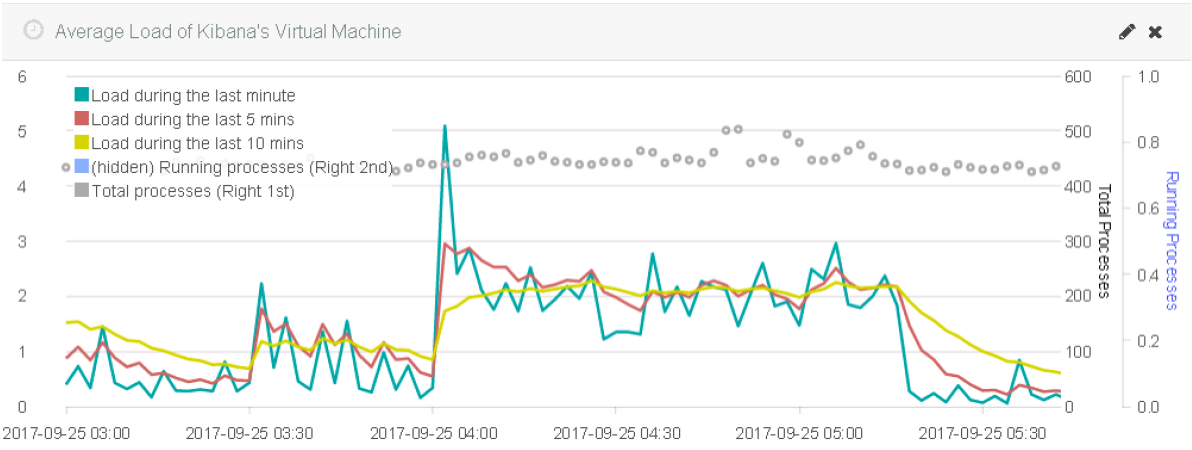



Figura 6.12: Estado de la máquina durante el cálculo de líneas base.

interés para la detección de anomalías se guardan junto con otra información relevante para la visualización de datos (v. gr. nombre del servicio, y otros metadatos).

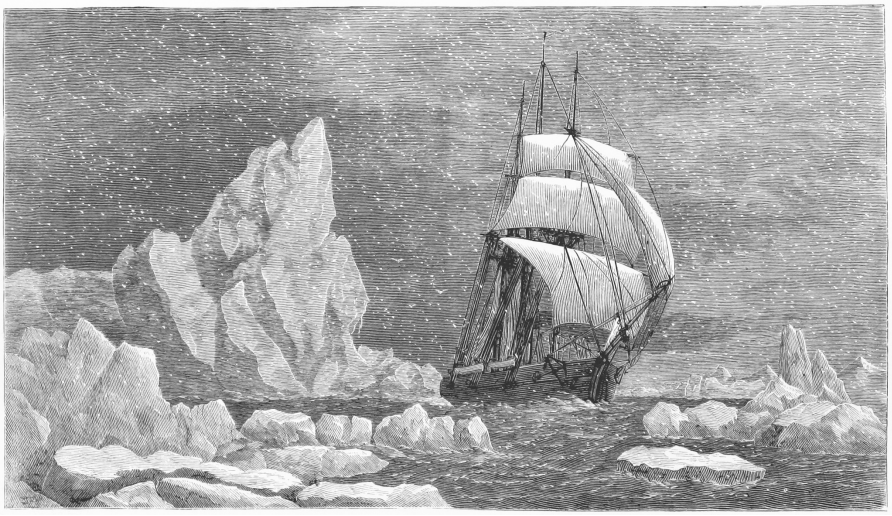
Por tanto, el rendimiento alcanzado resulta notable, aunque es necesario afirmar que mediante técnicas más sofisticadas se podrían mejorar los resultados obtenidos. No obstante, a la hora de desplegar este tipo de sistemas en entornos reales, se suele hacer más hincapié en las mejoras de requisitos funcionales (v. gr. facilidad de uso, visualización de alarmas, configuración), dejando de lado las mejoras no funcionales hasta que se requiera mayor rendimiento.

## 6.4. Conclusiones

En este capítulo se ha expuesto el proyecto de monitorización desplegado en la *Sociedad Estatal de Correos y Telégrafos* de España, como parte de un proyecto de *Naudit*. Este proyecto comenzó a desarrollarse en 2015 y continua en la actualidad. El trabajo realizado durante el doctorado se ha consolidado en varios proyectos en los que se ha aplicado el conocimiento adquirido a través de las distintas investigaciones y desarrollos realizados.

El conocimiento adquirido a través de la aplicación práctica de estas investigaciones resulta imprescindible a la hora de dirigir los esfuerzos de la investigación hacia problemas reales, desarrollando soluciones que satisfagan necesidades concretas del ámbito de la ingeniería informática, particularmente en el entorno de las redes de computadoras y la supervisión de sistemas.

Adicionalmente a la aplicación práctica de las metodologías para la detección de anomalías durante el proyecto, se pretende también transmitir la experiencia obtenida, consolidando las técnicas desarrolladas mediante la realización de publicaciones académicas con las que compartir el conocimiento adquirido con la comunidad.



*“Rescate de los supervivientes de la expedición Polaris por el Tigress.”*
1874

CAPÍTULO 7

# CONCLUSIONES

En este trabajo se han consolidado distintos estudios realizados en el campo de la telemática y las redes de computadoras. Durante estos años se han explorado diversas técnicas de computación de alto rendimiento para abordar los distintos retos encontrados en cada una de las etapas que conforman la cadena de procesos del análisis del tráfico.

Las investigaciones, progresos y aportaciones realizadas han repercutido en varios proyectos industriales, demostrando la utilidad que las contribuciones realizadas. Las aportaciones realizadas se reparten entre las siguientes áreas y etapas del proceso de monitorización y análisis de datos.

- Disección de tráfico multi-Gbps HTTP y reparto eficiente de la carga de trabajo.
- Recolección, almacenamiento y procesamiento de *logs* masivo.
- Análisis de datos enriquecidos para la generación de informes automáticos.
- El papel de las arquitecturas desagregadas en el análisis de datos.
- Metodologías para la detección de anomalías en el tráfico monitorizado.
- Métodos para la visualización de datos masivos.

## 7.1. Retos y principales contribuciones

Esta tesis se ha centrado en los distintos problemas presentes en el proceso de gestión de redes en grandes infraestructuras, más concretamente, en el análisis y monitorización del tráfico y los registros de eventos (*logs*) provenientes de servicios ejecutados en los distintos componentes de la red analizada. La creciente proliferación de servicios y el incremento de usuarios presentes en la red ha generado un aumento del tráfico a todos los niveles, requiriendo nuevas metodologías de alto rendimiento capaces de cubrir los actuales requisitos de velocidad y versatilidad para las etapas de recolección, disección, procesamiento, análisis y visualización de datos. A continuación se explican las distintas contribuciones realizadas y se resumen los métodos empleados para abordarlas.

### 7.1.1. Disección de tráfico multi-Gbps HTTP y reparto eficiente de la carga de trabajo.

El primero de los retos identificados consiste en la disección de tráfico HTTP a alta velocidad en *hardware* convencional para su fácil despliegue en entornos heterogéneos. Este ha sido abordado en el capítulo 2º, *HTTPanalyzer*: Disección del Tráfico HTTP, estudiando los diversos factores que afectan a la reconstrucción del flujo de información tras la captura del tráfico. En el caso estudiado, se ha contemplado el tráfico HTTP debido a su extenso uso en el desarrollo de servicios web y a su vital importancia para la medición tanto de la calidad de servicio como de la calidad de experiencia del usuario.

Se ha estudiado la importancia de los algoritmos para el reparto de la carga a procesar, proponiendo una alternativa que evita el reparto desigual de la carga producido por la presencia de conexiones con distinta cantidad de transacciones HTTP. Con todo esto se ha superado el rendimiento ofrecido por otras herramientas del estado del arte.

La solución desarrollada, *HTTPanalyzer*, ha sido empleada en numerosos proyectos reales aportando información crucial al analista durante el análisis del tráfico HTTP y la supervisión de los servicios web. Estos proyectos se han llevado a cabo gracias a la compañía *spin-off Naudit*, asociada al grupo de investigación *High Performance Computing and Networking* de la Universidad Autónoma de Madrid. Algunos de los clientes con los cuales se ha empleado esta solución son el *Banco Agromercantil de Guatemala*, *BBVA* y *Sabadell*, entre otros.

Como se verá en la sección 7.2, este trabajo ha sido publicado en la revista *Computer Networks*, compartiendo las lecciones aprendidas y retos encontrados con la comunidad investigadora y demás interesados en el área.

### 7.1.2. Recolección, almacenamiento y procesamiento de *logs*

El segundo de los retos abordados en esta tesis es la centralización de la recolección de mensajes de *log* en grandes infraestructuras de TI. Esta tarea debe hacerse con una gran proporcionalidad en el uso de los recursos, ya que, como es lógico, el proceso de recolección de información y monitorización debe requerir unos recursos varios órdenes de magnitud por debajo del número de nodos y componentes a monitorizar. De lo contrario, el sistema no escalaría adecuadamente en función del crecimiento de la infraestructura y requeriría una inversión enorme en términos de costes operacionales (CAPEX).

Por ello, en el capítulo 3º, *Loginson*: Recolección y Análisis de Registros, se ha propuesto un sistema de centralización de alto rendimiento para la recolección y procesamiento de *logs* a gran escala. A parte de la recolección de *logs*, también provee analíticas de alto nivel a través de una interfaz visual basada en Kibana con el fin de favorecer la resolución de incidentes críticos. La arquitectura simple de *Loginson*, prescinde de diversas funcionalidades presentes en las alternativas del estado del arte a cambio de un rendimiento por encima del otorgado por las soluciones del estado del arte. *Loginson* asegura un balanceo de los *logs* equilibrado gracias al reparto *Round-Robin* y un gran rendimiento en la escritura y lectura de los datos al utilizar ficheros planos con marcas temporales. Estas metodologías junto con otras técnicas aseguran un gran aprovechamiento vertical de los recursos.

El trabajo realizado durante el desarrollo de *Loginson*, así como los conceptos estudiados, han repercutido en el desarrollo de proyectos como el sistema de monitorización desplegado en la *Sociedad Estatal de Correos y Telégrafos* de España. Igualmente, este trabajo fue publicado en *The Journal of Supercomputing* en 2017, contribuyendo al conocimiento de este área.

### 7.1.3. Análisis de datos enriquecidos para la generación de informes automáticos

El tercero de los desafíos abordados es la generación de informes automáticos a partir de grandes cantidades de datos, ofreciendo a la vez versatilidad y velocidad para una fácil adaptación a nuevos escenarios. La automatización del análisis del tráfico en profundidad es un tema con poca presencia en la literatura sobre gestión de redes, principalmente por la dificultad para encontrar entornos heterogéneos en los que investigar. Generalmente estos entornos son empresariales y están cerrados al ámbito académico.

Para hacer frente a estos requisitos se han empleado lenguajes de alto nivel y también

desarrollado un módulo de lectura en C que ofrece una velocidad de lectura superior. De esta forma se ha mantenido una gran versatilidad sin sacrificar velocidad de procesamiento. Se ha conseguido el objetivo marcado de procesar los registros enriquecidos correspondientes a 1 TByte de trazas en menos de un cuarto de día, lo cual permite la generación de informes durante las horas menos cargadas (v. gr. la noche) para posteriormente desplegar medidas correctivas, en caso de necesitarlas. Los detalles sobre estas aportaciones se detallan en el capítulo 4º, *FERMIN*: Automatizando el Análisis del Tráfico.

Para un apropiado estudio de este problema ha sido fundamental contar con proyectos reales donde están involucrados grandes centros de datos. En este aspecto, *FERMIN* ha sido desarrollado gracias a la experiencia obtenida en distintos proyectos con *Naudit* en clientes como *BBVA*, *Ferrovial*, *Telefónica* o *Altamira*, entre otros. Este proyecto ha sido igualmente desplegado en estos mismos clientes, demostrando su funcionalidad.

El trabajo desarrollado está actualmente en proceso de envío en forma de artículo para revista bajo el título *Let the IT manager take a break: automatic traffic analysis from a hands-on perspective.*. Este trabajo también ha sido utilizado como caso de uso para otros trabajos en el ámbito de las arquitecturas desagregadas.

### 7.1.4. El papel de las arquitecturas desagregadas en el análisis de datos

Debido a que los centros de datos tradicionales están diseñados mediante una arquitectura monolítica, una gran porción de los recursos globales del centro de datos quedan ociosos debido a que el aprovisionamiento de los recursos se hace teniendo en mente posibles picos de utilización. Estos recursos quedan inutilizados durante los casos de utilización media.

Las arquitecturas desagregadas exploran la desagregación de los recursos con el fin de obtener una mayor flexibilidad, proporcionalidad en el uso de recursos y eficiencia energética. En el capítulo 5º, Diluyendo las Barreras de la Escalabilidad se explora la viabilidad de estas arquitecturas para tareas de análisis de datos donde la carga de trabajo es altamente variable. En concreto, se ha estudiado su viabilidad dentro de la propuesta del proyecto europeo *Disaggregated Recursive Datacentre-in-a-Box*.

Estas arquitecturas desagregadas son todavía objeto de estudio y desarrollo, y por ello, la investigación realizada, se ha centrado en el estudio del actual estado del desarrollo de estas arquitecturas junto con los distintos argumentos que las justifican y las implicaciones de estas arquitecturas en el ámbito del análisis de datos.

El trabajo realizado se presentará en Bangkok, Tailandia, en Diciembre de 2017 en la conferencia *IEEE International Conference on High Performance Computing and Communications* bajo el título *Diluting the Scalability Boundaries: Exploring the Use of Disaggregated Architectures for High-Level Network Data Analysis.* Allí se tendrá la oportunidad de debatir y discutir la viabilidad y capacidad de estas arquitecturas así como el futuro de las arquitecturas en grandes centros de datos.

### 7.1.5. Metodologías para la detección de anomalías en el tráfico monitorizado

Finalmente, en el capítulo 6º, Monitorización de redes se han puesto de manifiesto los retos encontrados para monitorizar, generar líneas base y detectar anomalías a partir de los datos recolectados, procesados y almacenados. En este capítulo se detalla el despliegue realizado en la *Sociedad Estatal de Correos y Telégrafos* de España y las diversas técnicas empleadas para una monitorización útil y proporcional con los recursos de la red en que se enmarca.

En este proyecto, se aúnan y describen las técnicas para recolección del tráfico y mensajes de *log* en un entorno real empresarial. Se ha conseguido realizar una monitorización de alarmas en tiempo real de más de 300 series temporales y la indexación de más de 7 millones de datos diarios mediante unos recursos acotados, además de la generación de *dashboards* para la monitorización reactiva de los sistemas gestionados a partir de estos datos.

En la actualidad este proyecto sigue creciendo adaptándose a nuevos servicios y recolectando nuevos datos de distintos orígenes (SNMP, bases de datos SCOM, etc.). Así mismo, el trabajo realizado se está recopilando para la elaboración de un artículo, con el título preliminar: *KISS methodologies for anomaly detection*. El conocimiento y experiencia ganado durante el despliegue de este proyecto será sin duda enriquecedor para la comunidad investigadora en el área de la gestión de redes.

### 7.1.6. Métodos para la visualización de datos masivos

No menos importante, y de forma paralela al desarrollo de los trabajos anteriores, se han estudiado diversas técnicas de visualización de datos para su aplicación en los trabajos realizados. Las distintas soluciones desarrolladas han tenido distintos requisitos de visualización de los datos. Por ejemplo, el sistema de informes automáticos ha necesitado diversas gráficas estáticas desarrolladas con distintas librerías (tikz, gnuplot, matplotlib,

etc.), que aun prescindiendo de la interactividad que ofrecen las soluciones web, sean capaces de mostrar información relevante, y de ser generados de forma automática. Por otro lado, los sistemas de monitorización mediante soluciones web han necesitado distintas soluciones web (kibana, plot.ly, entre otros) que cargan al cliente con el peso de la visualización y añaden un factor de interactividad a la visualización.

La importancia de los elementos de visualización ha sido esencial en todos los trabajos, ya que es la cara visible de los productos, soluciones y sistemas desarrollados, y generalmente su automatización va de la mano con una casuística muy amplia que requiere soluciones estables y bien adaptadas a los casos generales de uso. Generalmente, el desarrollo de estas visualizaciones está ligado con el método de consulta (*query workflow*) de los operadores de red sobre los datos, y la clase de preguntas y respuestas que esperan hacer al sistema de monitorización.

## 7.2. Contribuciones académicas

En esta sección se resumen los trabajos académicos elaborados durante el desarrollo del presente trabajo. Las contribuciones derivadas del trabajo realizado a lo largo de estos años se han ido consolidando en forma de artículos que describen las aportaciones y trabajos desarrollados. La tabla 7.1 relaciona las distintas publicaciones con los capítulos correspondientes.

### 7.2.1. Publicaciones como primer autor

#### 7.2.1.1. Publicaciones en revista

- Vega, C., Roquero, P., & Aracil, J. (2017). **Multi-Gbps HTTP traffic analysis in commodity hardware based on local knowledge of TCP streams.** Computer Networks, 113, 258–268.

  1er Tercil, Q1 y Q2 en ascenso, JCR 2016

  Índice de Impacto ultimos 5 años: 2.343

  Proyecto: Fed4Fire bajo los auspicios del European Union's Seventh Framework Programme (FP7)

- Vega, C., Roquero, P., Leira, R., Gonzalez, I., & Aracil, J. (2017). **Loginson: a transform and load system for very large-scale log analysis in large IT infrastructures.** The Journal of Supercomputing, 1-22.

2º Tercil, Q2, JCR 2016

Índice de Impacto ultimos 5 años: 1.349

Proyecto: TRÁFICA cofinanciado por el Ministerio de Economía y Competitividad (MINECO) y el Fondo Europeo de Desarrollo Regional (FEDER)

#### 7.2.1.2. Publicaciones en conferencia

- Vega, C., Zazo, J. F., Meyer, H., Zyulkyarov, F., Buedo, S. L., & Aracil, J. (2017). **Diluting the Scalability Boundaries: Exploring the Use of Disaggregated Architectures for High-Level Network Data Analysis.** IEEE International Conference on High Performance Computing and Communications in Bangkok, Thailand. 18-20 Dec.

  Ranking Core B.

  Índice de Aceptación: 33 %

  Proyecto: dReDBox apoyado por el programa European Union's Horizon 2020

- Vega, C. Buedo, S., Aracil, J. (2015) **EINSREPO Distributed Traffic Repository**. Poster. 2nd International Conference on Internet Science. Brussels, Belgium

  `http://research.cvega.es/docs/posters/poster_eins.pdf`

  Proyecto: Network of Excellence in Internet Science (EINS) bajo los auspicios del European Union's Seventh Framework Programme (FP7)

### 7.2.2. Otras publicaciones

- Perdices, D., de Vergara, J. E. L., Roquero, P., Vega, C., & Aracil, J. (2017). **FlexiTop: sistema escalable y flexible de medidas de calidad para servicios Over-The-Top. XIII Jornadas de Ingeniería Telemática.** Valencia, Spain.

  Proyecto: TRÁFICA cofinanciado por el Ministerio de Economía y Competitividad (MINECO) y el Fondo Europeo de Desarrollo Regional (FEDER)

- Leira, R., Roquero, P., Vega, C., González, I., & Aracil, J. (2016) **HPSEngine: Motor de alto rendimiento y baja latencia para el procesamiento distribuido en tiempo real.** Jornadas Sarteco 2016. Salamanca, Spain.

### 7.2.3. Publicaciones en producción

- Vega, C., Miravalls, E., Julián-Moreno, G., de Vergara, J. E. L., Magaña, E., & Aracil, J. **Let the IT manager take a break: automatic traffic analysis from a hands-on perspective.**
- Vega, C., & Aracil, J. **KISS methodologies for anomaly detection.**

Cuadro 7.1: Relación de capítulos y trabajos realizados

| **Capítulo** | **Título del artículo, proyecto o trabajo** |
|---|---|
| 1º: Introducción | Motivación, objetivos, estructura del trabajo. |
| 2º: *HTTPanalyzer*: Disección del Tráfico HTTP | Multi-Gbps HTTP traffic analysis in commodity hardware based on local knowledge of TCP streams. |
| 3º: *Loginson*: Recolección y Análisis de Registros | Loginson: a transform and load system for very large-scale log analysis in large IT infrastructures |
| 4º: *FERMIN*: Automatizando el Análisis del Tráfico | Let the IT manager take a break: automatic traffic analysis from a hands-on perspective. Proyecto FERMIN |
| 5º: Diluyendo las Barreras de la Escalabilidad | Diluting the Scalability Boundaries: Exploring the Use of Disaggregated Architectures for High-Level Network Data Analysis |
| 6º: Monitorización de redes | KISS methodologies for anomaly detection. Proyecto de monitorización en la Sociedad Estatal de Correos y Telégrafos de España |
| 7º: Conclusiones | Resumen de las contribuciones, trabajo futuro. |

# BIBLIOGRAFÍA


[1] Gifreu, Josep. *Ibertex será la denominación del servicio español de videotex.* (1983) El País
https://elpais.com/diario/1983/02/08/radiotv/413506804_850215.html

[2] Metcalfe, B. (2013). *Metcalfe's law after 40 years of ethernet*. IEEE Computer Society, 46(12), 26-31.
https://doi.org/10.1109/MC.2013.374

[3] Hendler, J., and Golbeck, J. (2008). *Metcalfe's law, Web 2.0, and the Semantic Web*. Web Semantics: Science, Services and Agents on the World Wide Web, 6(1), 14-20.

[4] Cisco Visual Networking Index: Forecast and Methodology, 2016–2021, white paper.
https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[5] Julián Moreno, G. (2016). *Monitorización, captura y almacenamiento inteligente de tráfico de red a 40Gbps.* Universidad Autónoma de Madrid
http://hdl.handle.net/10486/675469

[6] Vega Moreno, C. G. (2014). *Disección de tráfico web a alta velocidad.* Universidad Autónoma de Madrid
http://hdl.handle.net/10486/663649

[7] Peña-López, Ismael and others: *OECD Internet Economy Outlook 2012*, Chapter 4 (2012)
http://dx.doi.org/10.1787/9789264086463-en

[8] Stas Khirman and Peter Henriksen: *Relationship between Quality-of-Service and Quality-of-Experience for Public Internet Service*. In Proc. of the 3rd Workshop on Passive and Active Measurement. (2002)
http://www-v1.icir.org/2002/Relationship_Between_QoS_and_QoE.pdf

[9] García-Dorado, José Luis and Santiago del Río, Pedro M and Ramos, Javier and Muelas, David and Moreno, Victor and López de Vergara, Jorge E and Aracil, Javier. *Low-cost and high-performance: VoIP monitoring and full-data retention at multi-Gb/s rates using commodity hardware*. International Journal of Network Manage-

ment. Vol. 24. (no. 3) 181-199. (2014)
http://dx.doi.org/10.1002/nem.1858

[10] Postel, J. (2003). RFC 793: Transmission control protocol, September 1981.
https://tools.ietf.org/html/rfc793#section-1.5 (Accessed: 20 Ago. 2017)

[11] Jing Xu; Hanbo Wang; Wei Liu; Xiaojun Hei: *Towards High-Speed Real-Time HTTP Traffic Analysis on the Tilera Many-Core Platform*. IEEE HPCC_EUC, (2013)
http://dx.doi.org/10.1109/HPCC.and.EUC.2013.252

[12] Kai Zhang; Junchang Wang; Bei Hua; Xinan Tang: *Building High-performance Application Protocol Parsers on Multi-core Architectures*. IEEE 17th ICPADS. (2011)
http://dx.doi.org/10.1109/ICPADS.2011.37

[13] Bro.org: *The Bro Network Security Monitor*. (2013)
http://www.bro.org (Accessed: 20 Ago. 2017)

[14] Vern Paxson, *Bro: A System for Detecting Network Intruders in Real-Time*.
International Journal of Computer and Telecommunications Networking (1998)
https://doi.org/10.1016/S1389-1286(99)00112-7

[15] Eric Lawrence: *Debugging with Fiddler*. (2012)
https://fiddlerbook.com/book/ (Accessed: 20 Ago. 2017)

[16] Mosberger, D., and Jin, T. (1998). httperf — A tool for measuring web server performance.
ACM SIGMETRICS Performance Evaluation Review, 26(3), 31-37.
https://doi.org/10.1145/306225.306235

[17] NetFlow, Cisco IOS, White Paper, *Introduction to Cisco IOS NetFlow-A Technical Overview* (2006)
http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

[18] Plonka, David, in LISA, pp. 305-317, *FlowScan: A Network Traffic Flow Reporting and Visualization Tool* (2000)
https://www.usenix.org/legacy/events/lisa2000/full_papers/plonka/plonka.pdf

[19] Rafal Wojtczuk: *Libnids, an implementation of an E-component of Network Intrusion Detection System*. (2010)
http://libnids.sourceforge.net/ (Accessed: 20 Ago. 2017)

[20] Orebaugh, A., Ramirez, G., and Beale, J. (2006). Wireshark and Ethereal network protocol analyzer toolkit. Syngress.

[21] Combs, G. (2007). Wireshark. http://www.wireshark.org/ (Accessed: 20 Ago. 2016)

[22] McCanne, S., and Jacobson, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In USENIX winter (Vol. 46). (1993)
http://dl.acm.org/citation.cfm?id=1267305

[23] Korth, H. F., and Silberschatz, A : *Database system concepts 6th Edition*. Chapter 11, page 510. (2010)

[24] García-Dorado, J. L., Hernández, J. A., Aracil, J., de Vergara, J. E. L., Monserrat, F. J., Robles, E., and de Miguel, T. P: *On the duration and spatial characteristics of Internet traffic measurement experiments*.
IEEE Communications Magazine, Vol.46 issue 11, 148-155. (2008)
http://dx.doi.org/10.1109/MCOM.2008.4689258

[25] Shi, W., MacGregor, M. H., and Gburzynski, P. *An adaptive load balancer for multiprocessor routers*. Simulation, Vol. 82 (no. 3), 173-192. (2006)
http://dx.doi.org/10.1177/0037549706067079

[26] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. *RFC 2616 Hypertext Transfer Protocol - HTTP/1.1 Hypertext Transfer Protocol.* (1999)
http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.2.1

[27] *Boutell.com: What is the maximum length of a URL?*. (2006)
http://www.boutell.com/newfaq/misc/urllength.html (Accessed: 20/08/2017)

[28] Sitemaps.org: *Sitemaps XML format.* (2008)
http://www.sitemaps.org/protocol.html (Accessed: 20 Ago. 2017)

[29] Jose Fernando Zazo, Marco Forconesi, Sergio Lopez-Buedo, Gustavo Sutter, and Javier Aracil: *TNT10G: A high-accuracy 10 GbE traffic player and recorder for multi-Terabyte traces*. ReConFig14. (2014)
http://dx.doi.org/10.1109/ReConFig.2014.7032561

[30] José Luis García-Dorado and Jose Fernando Zazo, HPCN-UAM: *10Gbps Trace Reproduction testbed for testing software-defined networks (10GTRACE-TESTER)*. (2015)
http://www.fed4fire.eu/10g-trace-tester/ (Accessed: 20 Ago. 2017)

[31] Victor Moreno, Pedro M. Santiago del Río, Javier Ramos, David Muelas, José Luis García-Dorado, Francisco J Gomez-Arribas, and Javier Aracil:
*Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems*.
International Journal of Network Management (2014)
http://dx.doi.org/10.1002/nem.1861

[32] Vega, Carlos., Roquero, Paula., and Aracil, Javier. (2017). Multi-Gbps HTTP traffic analysis in commodity hardware based on local knowledge of TCP streams. Computer Networks, 113, 258-268.
http://doi.org/10.1016/j.comnet.2017.01.001

[33] Chaudhuri, S., Dayal, U., and Narasayya, V. (2011). An overview of business intelligence technology. *Communications of the ACM*, 54(8), 88-98. https://doi.org/10.1145/1978542.1978562

[34] White, T. (2012). Hadoop: The definitive guide. *O'Reilly Media, Inc.*

[35] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010, May). The hadoop distributed file system. In *Mass storage systems and technologies (MSST)*, 2010 IEEE 26th symposium on (pp. 1-10). IEEE. http://dx.doi.org/10.1109/MSST.2010.5496972

[36] George, L. (2011). HBase: The Definitive Guide: Random Access to Your Planet-Size Data. *O'Reilly Media, Inc.*

[37] Kuć, R., and Rogoziński, M. (2015). Mastering Elasticsearch. Packt Publishing Ltd.

[38] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M. and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2), 4. https://doi.org/10.1145/1365815.1365816

[39] Peña-López, Ismael and others: *OECD Internet Economy Outlook 2012*, Chapter 4 (2012) http://dx.doi.org/10.1787/9789264086463-en

[40] Networking, C. V. (2016). *Cisco Global Cloud Index*: Forecast and Methodology, 2015-2020. White paper. http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf (Accessed: 20 Ago. 2017)

[41] Snyder, J. (2007). Microsoft: datacenter growth defies Moore's law. *PC-World*. http://www.pcworld.com/article/130921/article.html (Accessed: 20/08/17)

[42] Ashton, K. (2009). That 'internet of things' thing. *RFiD Journal*, 22(7), 97-114.

[43] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7), 1645-1660. http://dx.doi.org/10.1016/j.future.2013.01.010

[44] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15), 2787-2805. http://dx.doi.org/10.1016/j.comnet.2010.05.010

[45] Gerhards, R. (2009). RFC 5424: The *syslog* protocol. https://tools.ietf.org/html/rfc5424 (Accessed: 20 Ago. 2017)

[46] Yang, Fangjin and Tschetter, Eric and Léauté, Xavier and Ray, Nelson and Merlino, Gian and Ganguli, Deep. *Druid: a real-time analytical data store*. Proceedings of the 2014 ACM SIGMOD international conference on Management of data

[47] Jacobs, A. (2009). The pathologies of big data. *Communications of the ACM*, 52(8), 36-44. https://doi.org/10.1145/1536616.1536632

[48] Azarmi, B. (2016). The Big (Data) Problem. In Scalable Big Data Architecture (pp. 1-16). http://dx.doi.org/10.1007/978-1-4842-1326-1_1

[49] C. Kalantzis. Revisiting 1 million writes per second. 2014 http://techblog.netflix.com/2014/07/revisiting-1-million-writes-per-second.html (Accessed: 20 Ago. 2017)

[50] BalaBit. How to configure *syslog-ng* pe to cooperate with splunk. (2015) https://www.balabit.com/documents/pdf/syslog-ng-pe-whitepaper-splunk.pdf (Accessed: 20 Ago. 2017)

[51] Scylla. *Scylla vs. cassandra* benchmark. (2015) http://www.scylladb.com/technology/scylla-vs-cassandra-benchmark/ (Accessed: 20 Ago. 2017) http://www.scylladb.com/2015/10/09/single-node-benchmark-1-8m-tps/ (Accessed: 20 Ago. 2017)

[52] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, et al. Achieving 100,000,000 database inserts per second using accumulo and d4m. In *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, pages 1–6. IEEE, 2014.

[53] Kreps, J., Narkhede, N., and Rao, J. (2011, June). Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB (pp. 1-7).

[54] J. Kreps. *Benchmarking Apache Kafka*: 2 million writes per second (on three cheap machines). (2014) https://engineering.linkedin.com/kafka/benchmarking%2dapache%2dkafka%2d2%2dmillion%2dwrites%2dsecond%2dthree%2dcheap%2dmachines (Accessed: 20 Ago. 2017)

[55] H. Shreedharan. *Using Flume: Flexible, Scalable, and Reliable Data Streaming*. O'Reilly Media, Inc., 2014.

[56] E. Silva. *Fluentd*: a high performance unified logging layer. (2015) https://www.linux.com/news/fluentd-high-performance-unified-logging-layer (Accessed: 20 Ago. 2017)

[57] E. Silva. Unifying events & logs into the cloud. (2015)
http://events.linuxfoundation.org/sites/events/files/slides/unifying_events.pdf (Accessed: 20 Ago. 2017)

[58] FluentD. *Fluentd architecture*. (2015)
http://www.fluentd.org/architecture (Accessed: 20 Ago. 2017)

[59] E. POINT. Benchmarking top noSQL databases. (2014)
http://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf (Accessed: 20 Ago. 2017)

[60] S. Corp. On-disk microbenchmark. (2014)
http://symas.com/mdb/ondisk/ (Accessed: 10 Jul. 2016)
http://archive.is/GWaVx (Accessed: 20 Ago. 2017)

[61] *Scylla architecture*. (2015)
http://www.scylladb.com/technology/architecture/ (Accessed: 20 Ago. 2017)

[62] *Scylla vs. Cassandra* benchmark (cluster). (2015)
http://www.scylladb.com/technology/scylla-vs-cassandra-benchmark-cluster/ (Accessed: 10/7/16)
http://www.scylladb.com/2015/10/13/cluster-benchmark/ (Accessed: 2/8/17)

[63] R. Bodkin. *Mapr* releases commercial distributions based on hadoop. (2011)
http://www.infoq.com/news/2011/07/mapr (Accessed: 20 Ago. 2017)

[64] J. Scott. Loading a time series database at 100 million points per second.
https://www.mapr.com/blog/loading-time-series-database-100-million-points-second (Accessed: 20 Ago. 2017)

[65] *Splunk*. Splunkit v2.0.2 results and ec2 storage comparisons. (2013)
http://blogs.splunk.com/2013/06/06/splunkit-v2-0-2-results-ec2-storage-comparisons/ (Accessed: 20 Ago. 2017)

[66] Splunk. *Splunk sizing and performance*: Doing more with more. (2014)
http://blogs.splunk.com/2014/05/07/splunk-sizing-and-performance-doing-more-with-more/ (Accessed: 20 Ago. 2017)

[67] Elasticsearch nightly benchmarks (2016)
https://benchmarks.elastic.co/index.html
https://elasticsearch-benchmarks.elastic.co/nested/index.html
(Accessed: 20 Ago. 2017)

[68] B. SA. Sending messages directly to *Elasticsearch*. (2015)
https://www.balabit.com/sites/default/files/documents/syslog-ng-ose-latest-guides/en/syslog-ng-ose-guide-admin/html/configuring-destinations-elasticsearch.html (Accessed: 20 Ago. 2017)

[69] Vega, C., Roquero, P., Leira, R., Gonzalez, I., and Aracil, J. (2017). Loginson: a transform and load system for very large-scale log analysis in large IT infrastructures. The Journal of Supercomputing, 1-22. https://dx.doi.org/10.1007/s11227-017-1990-1

[70] Cost of datacenter Outages, datacenter Performance Benchmark Series, Ponemon Institute, January 2016. http://www.ponemon.org/blog/2016-cost-of-data-center-outages

[71] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating events with time series for incident diagnosis. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '14). ACM, New York, NY, USA, 1583-1592. http://doi.org/10.1145/2623330.2623374

[72] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M. (2007, August). Towards highly reliable enterprise network services via inference of multi-level dependencies. In ACM SIGCOMM Computer Communication Review (Vol. 37, No. 4, pp. 13-24). ACM. http://doi.org/10.1145/1282380.1282383

[73] Ciraci, S., Loo, B. T., Schuster, A., and Outhred, G. (2016, August). Taking the Blame Game out of datacenters Operations with NetPoirot. In Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference (pp. 440-453). ACM. http://doi.org/10.1145/2934872.2934884

[74] Cisco Global Cloud Index: Forecast and Methodology, 2015–2020 http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf

[75] Goodall, J. R., Lutters, W. G., Rheingans, P., and Komlodi, A. (2005, October). Preserving the big picture: Visual network traffic analysis with tnv. In Visualization for Computer Security, 2005.(VizSEC 05). IEEE Workshop on (pp. 47-54). IEEE. http://doi.org/10.1109/VIZSEC.2005.1532065

[76] Shi, W., MacGregor, M. H., and Gburzynski, P. (2005). Load balancing for parallel forwarding. IEEE/ACM Transactions on Networking (TON), 13(4), 790-801. https://doi.org/10.1109/TNET.2005.852881

[77] Deri, L., Martinelli, M., Bujlow, T., and Cardigliano, A. (2014, August). ndpi: Open-source high-speed deep packet inspection. In Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International (pp. 617-622). IEEE. http://doi.org/10.1109/IWCMC.2014.6906427

[78] Robin Sommer and Anja Feldmann. 2002. NetFlow: information loss or win?. In Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment (IMW '02). ACM, New York, NY, USA, 173-174. http://dx.doi.org/10.1145/637201.637226

[79] Schlosser, D., Staehle, B., Binzenhofer, A., and Boder, B. (2010, May). Improving the QoE of citrix thin client users. In Communications (ICC), 2010 IEEE International Conference on (pp. 1-6). IEEE. http://doi.org/10.1109/ICC.2010.5501891

[80] Rougier, N. P., Droettboom, M., and Bourne, P. E. (2014). Ten simple rules for better figures. PLoS computational biology, 10(9), e1003833. http://doi.org/10.1371/journal.pcbi.1003833

[81] Totoni, E., Anderson, T. A., and Shpeisman, T. (2017). *HPAT: high performance analytics with scripting ease-of-use.* Proceedings of the International Conference on Supercomputing.
http://dx.doi.org/10.1145/3079079.3079099

[82] Shehabi, A., Smith, S. J., Horner, N., Azevedo, I., Brown, R., Koomey, J., and Lintner, W. (2016). United States data center energy usage report. Lawrence Berkeley National Laboratory, Berkeley, California. LBNL-1005775 Page, 4. https://eta.lbl.gov/publications/united-states-data-center-energy

[83] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys and Tutorials, 17(4), 2347-2376. https://doi.org/10.1109/COMST.2015.2444095

[84] Manyika, J., Chui, M., Bughin, J., Dobbs, R., Bisson, P., and Marrs, A. (2013). Disruptive technologies: Advances that will transform life, business, and the global economy (Vol. 180). San Francisco, CA: McKinsey Global Institute. http://library.wur.nl/WebQuery/clc/2079131

[85] Arista in Q1 2017 https://s2.q4cdn.com/209832288/files/doc_presentations/2017/q1/2017-Highlights-Q1.pdf

[86] Zazo, J. F., Lopez-Buedo, S., Sutter, G., and Aracil, J. (2016, November). Automated synthesis of FPGA-based packet filters for 100 Gbps network monitoring applications. In ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on (pp. 1-6). IEEE. https://doi.org/10.1109/ReConFig.2016.7857156

[87] Pagès, A., Serrano, R., Perelló, J., and Spadaro, S. (2017). On the benefits of resource disaggregation for virtual data centre provisioning in optical data centres. Computer Communications, 107, 60-74. https://doi.org/10.1016/j.comcom.2017.03.009

[88] Lim, K., Chang, J., Mudge, T., Ranganathan, P., Reinhardt, S. K., and Wenisch, T. F. (2009, June). Disaggregated memory for expansion and sharing in blade servers. In ACM SIGARCH Computer Architecture News (Vol. 37, No. 3, pp. 267-278). ACM. https://doi.org/10.1145/1555754.1555789

[89] Tu, C. C., Lee, C. T., and Chiueh, T. C. (2014, October). Marlin: A memory-based rack area network. In Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems (pp. 125-136). ACM. https://doi.org/10.1145/2658260.2658262

[90] Han, S., Egi, N., Panda, A., Ratnasamy, S., Shi, G., and Shenker, S. (2013, November). Network support for resource disaggregation in next-generation datacenters. In Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks (p. 10). ACM. https://doi.org/10.1145/2535771.2535778

[91] Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2008, August). Towards a next generation data center architecture: scalability and commoditization. In Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow (pp. 57-62). ACM. https://doi.org/10.1145/1397718.1397732

[92] Kachris, C., and Tomkos, I. (2012). A survey on optical interconnects for data centers. IEEE Communications Surveys and Tutorials, 14(4), 1021-1036. https://doi.org/10.1109/SURV.2011.122111.00069

[93] Fiorani, M., Aleksic, S., Casoni, M., Wosinska, L., and Chen, J. (2014). Energy-efficient elastic optical interconnect architecture for data centers. IEEE Communications Letters, 18(9), 1531-1534. https://doi.org/10.1109/LCOMM.2014.2339322

[94] Abali, B., Eickemeyer, R. J., Franke, H., Li, C. S., and Taubenblatt, M. A. (2015). Disaggregated and optically interconnected memory: when will it be cost effective?. https://arxiv.org/abs/1503.01416

[95] Hasharoni, K. (2014, July). High BW parallel optical interconnects. In Photonics in Switching (pp. PT4B-1). Optical Society of America. https://doi.org/10.1364/PS.2014.PT4B.1

[96] Hasharoni, K., Benjamin, S., Geron, A., Stepanov, S., Katz, G., Epstein, I., and Mesh, M. (2014, March). A 1.3 Tb/s parallel optics VCSEL link. In SPIE OPTO (pp. 89910C-89910C). International Society for Optics and Photonics. http://doi.org/10.1117/12.2038073

[97] C. Reiss, J. Wilkes, J.L. Hellerstein, Google Cluster-Usage Traces: Format+Schema, 2012 Google Technical Report http://code.google.com/p/googleclusterdata/wiki/TraceVersion2

[98] Samih, A., Wang, R., Maciocco, C., Tai, T. Y. C., and Solihin, Y. (2011, October). A collaborative memory system for high-performance and cost-effective clustered architectures. In Proceedings of the 1st Workshop on Architectures and Systems for Big Data (pp. 4-12). ACM. https://doi.org/10.1145/2377978.2377979

[99] Yan, G., Ma, J., Han, Y., and Li, X. (2016). EcoUp: Towards Economical Datacenter Upgrading. IEEE Transactions on Parallel and Distributed Systems, 27(7), 1968-1981. https://doi.org/10.1109/TPDS.2015.2477827

[100] dReDBox Deliverable: Requirements specification and KPIs Document https://s3-eu-west-1.amazonaws.com/dredbox/Deliverables/dRedBox_Deliverable_D2.1_REVISION.pdf

[101] Li, C. S., Franke, H., Parris, C., Abali, B., Kesavan, M., and Chang, V. (2017). Composable architecture for rack scale big data computing. Future Generation Computer Systems, 67, 180-193. https://doi.org/10.1016/j.future.2016.07.014

[102] Sales in U.S. Department Stores during the period 2000-2016 https://www.census.gov/econ/currentdata/dbsearch?program=MRTS&startYear=2000&endYear=2016&categories=4521I&dataType=MPCSM&geoLevel=US&notAdjusted=1&submit=GET+DATA&releaseScheduleId=

[103] The 2015 Holiday Season. U.S. Census CB15-FF.25 https://goo.gl/ywK5eK

[104] Katrinis, K., Zervas, G., Pnevmatikatos, D., Syrivelis, D., Alexoudi, T., Theodoropoulos, D., and Chen, Q. (2016, June). On interconnecting and orchestrating components in disaggregated data centers: The dReDBox project vision. In Networks and Communications (EuCNC), 2016 European Conference on (pp. 235-239). IEEE. https://doi.org/10.1109/EuCNC.2016.7561039

[105] Grisenthwaite, R. (2011). ARMv8 technology preview. In IEEE Conference. https://goo.gl/Bm2uL2

[106] ARM Annual Report, 2015.

[107] Rajovic, N., Carpenter, P. M., Gelado, I., Puzovic, N., Ramirez, A., and Valero, M. (2013, November). Supercomputing with commodity CPUs: Are mobile SoCs ready for HPC?. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (p. 40). ACM. https://doi.org/10.1145/2503210.2503281

[108] Durand, Y., Carpenter, P. M., Adami, S., Bilas, A., Dutoit, D., Farcy, A., and Matus, E. (2014, August). Euroserver: Energy efficient node for european micro-servers. In Digital System Design (DSD), 2014 17th Euromicro Conference on (pp. 206-213). IEEE. https://doi.org/10.1109/DSD.2014.15

[109] Hugo Meyer, Jose Carlos Sancho, Josue V. Quiroga, Ferad Zyulkyarov, Damian Roca and Mario Nemirovsky.: *Disaggregated Computing. An Evaluation of Current Trends for Datacentres*. International Conference on Computational Science (ICCS) 2017 http://doi.org/10.1016/j.procs.2017.05.129

[110] Guelbenzu de Villota, G., Calabretta, N., and Raz, O. (2015, November). Mid-board optics as an essential building block for future data center switches. In Proceedings of the 20th Annual Symposium of the IEEE Photonics Benelux Chapter (Vol. 26, p. 27). https://pure.tue.nl/ws/files/19976085/mid.pdf

[111] Natarajan, A., Ning, P., Liu, Y., Jajodia, S., and Hutchinson, S. E. (2012). NSDMiner: Automated discovery of network service dependencies (pp. 2507-2515). IEEE INFOCOM. https://doi.org/10.1109/INFCOM.2012.6195642

[112] Chen, X., Zhang, M., Mao, Z. M., and Bahl, P. (2008, December). Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions. In OSDI (Vol. 8, pp. 117-130). https://www.usenix.org/legacy/event/osdi08/tech/full_papers/chen_xu/chen_xu_html/

[113] Dunning, T., and Ertl, O. (2014). Computing Extremely Accurate Quantiles Using t-Digests. github. com. https://raw.githubusercontent.com/tdunning/t-digest/master/docs/t-digest-paper/histo.pdf

[114] Muelas, D., Gordo, M., García-Dorado, J. L., and de Vergara, J. E. L. (2015, November). Dictyogram: A statistical approach for the definition and visualization of network flow categories. In Network and Service Management (CNSM), 2015 11th International Conference on (pp. 219-227). IEEE. https://doi.org/10.1109/CNSM.2015.7367362

[115] Campos, G. O., Zimek, A., Sander, J., Campello, R. J., Micenková, B., Schubert, E., and Houle, M. E. (2016). On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. Data Mining and Knowledge Discovery, 30(4), 891-927. https://doi.org/10.1007/s10618-015-0444-8

[116] Ramaswamy, S., Rastogi, R., and Shim, K. (2000, May). Efficient algorithms for mining outliers from large data sets. In ACM Sigmod Record (Vol. 29, No. 2, pp. 427-438). ACM.
http://dl.acm.org/citation.cfm?id=335437

[117] Zhang, X., Meng, F., Chen, P., and Xu, J. (2016, June). TaskInsight: A fine-grained performance anomaly detection and problem locating system. (2016) IEEE 9th International Conference on Cloud Computing (CLOUD) (pp. 917-920).
https://doi.org/10.1109/CLOUD.2016.0136

[118] Calheiros, R. N., Ramamohanarao, K., Buyya, R., Leckie, C., and Versteeg, S. (2017). On the effectiveness of isolation–based anomaly detection in cloud data centers. Concurrency and Computation: Practice and Experience.
http://dx.doi.org/10.1002/cpe.4169

[119] Liu, F. T., Ting, K. M., and Zhou, Z. H. (2008, December). Isolation forest. ICDM'08. Proceedings of the 8th IEEE International Conference on Data Mining (pp. 413-422).
https://doi.org/10.1109/ICDM.2008.17

[120] Bateman, S., Mandryk, R. L., Gutwin, C., Genest, A., McDine, D., and Brooks, C. (2010, April). Useful junk?: the effects of visual embellishment on comprehension and memorability of charts. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 2573-2582). ACM.
http://dl.acm.org/citation.cfm?id=1753716